

Survival Kit: a Constraint-Based Behavioural Architecture for Robot Navigation

Pedro Santana¹ and Luís Correia²

¹ IntRoSys S.A.

Quinta da Torre, Campus FCT-UNL
2829-516 - Portugal

² University of Lisbon,
Campo Grande
1749-016 – Portugal

Abstract. This article presents a constraint-based behavioural architecture for low-level safe navigation, the Survival Kit. Instead of approaching the problem by customising a generic Behaviour-Based architecture, the Survival Kit embodies a dedicated semantics for safe navigation, which augments its expressiveness for the task. An instantiation of the architecture for goal-oriented obstacle avoidance in unstructured indoor environments is proposed. Special attention is given to an environmental feature, the gap, which allows to optimise paths based on immediate ranging data. Experimental results in simulation confirm the capabilities of the approach.

1 Introduction

The motivation for this work is the development of control systems for disposable robots to be applied in hazardous tasks. In those tasks, global localisation and communication mechanisms are reduced or even absent; in addition, robots may get lost or damaged. Another driving force towards the development of simple robot control architectures is the increasing interest on micro and nano-robots. In this sense, the main set of requirements for this work is: the control system should rely as little as possible on localisation mechanisms, complex sensory apparatus, and it should target implementations for simple computational units (e.g. micro-controllers).

Previous work on Behaviour-Based architectures, such as those based on priority (e.g. [2, 3]) and action-selection coordination (e.g. [6]) mechanisms either are extremely complex or fail to produce smooth and optimised trajectories. Since only a single behaviour is active at a time, either it includes other skills increasing complexity and reducing modularity, or it will not be goal-oriented (e.g. avoiding obstacles without considering which direction would also benefit a goal seeking behaviour). In voting- (e.g. [8]) and fusion-based (e.g. [1]) Behaviour-Based architectures, the produced action may not satisfy all constituent behaviours, which may reduce the robustness of the system, in case

those behaviours are responsible for maintaining robot’s safety. In addition, (tedious) tuning of each behaviour contribution to the overall behaviour is usually required, which often compels to choose between robustness and path smoothing.

Typically, Behaviour-Based architectures are tackled in a holistic perspective, where the same semantics is used in all parts of the control system, such as safety keeping and task-achieving parts. This paper approaches the problem differently, it assumes that different semantics are required at each level. In this sense, this work introduces an architecture, the Survival Kit (SK) architecture, whose semantics is specially designed for safe navigation, delegating task-achieving requirements to upper-layers, which can be implemented by any other architecture (not necessarily a Behaviour-Based one). The coordination mechanism implemented in the SK architecture is based on constraints, which are added by several reflexes running in parallel, according to their skills. This type of *cooperation by negation* guarantees that the resulting action does not disagree with any of the reflexes. Although this feature is essential for safe navigation, it may be restrictive for task-achieving behaviours.

As it will be shown, the semantics of the SK will allow the implementation of a simple yet robust goal-oriented obstacle avoidance method, which taking into consideration the target implementation, i.e. disposable robots, is capable of competing with the most popular methods, such as *Dynamic Window Approach* [4], *Curvature Velocity Method* [9], *VFH+* [11], *Nearness Diagram* [7], and *Potential Fields* based approaches [1]. Since the obstacle avoidance method herein proposed is supported by a behavioural architectural framework, it is potentially more generalisable and modular than the previously referred algorithmic based approaches.

2 The Survival Kit Architecture

The SK architecture intends to be used as the bottom layer of a robot control system, providing it with safe navigation capabilities. Thus, everything required to maintain the survival (in terms of immediate reactions) of the robot should be implemented within the SK architecture. The constrained application of this architecture to safe navigation allows the definition of a well adapted semantics to the problem at hand, avoiding to lose time customising a generic architecture. Relying on the SK to guarantee safe navigation, upper decision/control layers are relieved of handling real-time events.

On the other hand, a too restrictive SK architecture would fail in its applicability; therefore, in order to introduce some plasticity to the SK, upper-layers are allowed to modulate the SK activity. In addition, the SK architecture has been designed to disturb as less as possible the task-achievement of upper-layers.

Figure 1 illustrates the main components of the SK architecture. Briefly, a set of reflexes **inhibits** some of the available action features according to sensory information. Then, according to a given criteria, the best action is selected and sent to the actuators. Upper layers can **modulate** the SK by inhibiting action

features, as reflexes do, and by defining which motor action pattern is preferred (e.g. *desired heading* and *desired linear velocity*).

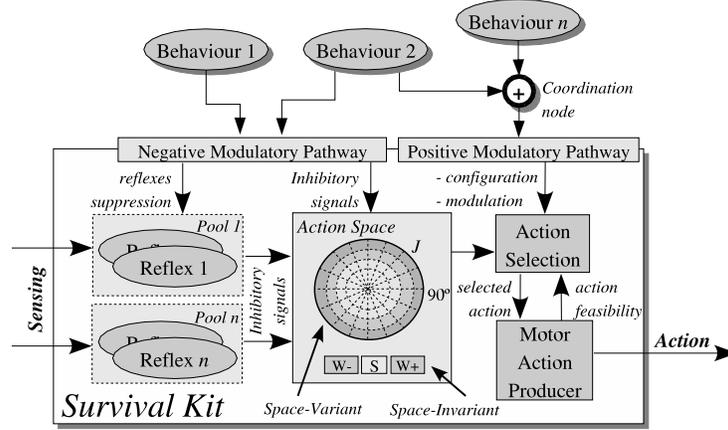


Fig. 1. The Survival Kit Architecture. Behaviours are exemplifying a task-achieving upper-layer.

2.1 The Action Feature Space

The core of the architecture is the *action feature space*, which describes indirectly all available actions to the robot. An example of an action feature is v_{max} , which stands for *the maximum linear velocity allowed for a given sector of the environment*. The *action feature space* is composed of two sub-spaces: the *space-variant* and the *space-invariant*.

Space-Variant Action Feature Sub-Space The *space-variant action feature sub-space* is sectorial (see figure 1). Each sector $j \in J$ with an angular size ϕ , where J is the set of all possible sectors, corresponds to a region in the environment with the same shape. This design is motivated by the fact that most sensory information is radial; thus, a similar representation facilitates further processing. A set of *space-variant action feature descriptors* F_{sv} must be defined every time the SK architecture is instantiated. In each sector, associated to each feature descriptor $f_{sv} \in F_{sv}$, there are two slots, one for a constraint c on the respective action feature, and another one for its temporal validity τ .

Setting a constraint c (e.g. to 1 ms^{-1}) and its corresponding τ (e.g. to 100 ms) associated to a feature descriptor f_{sv} (e.g. v_{max}), in a given sector j (e.g. 2), will affect the set of available actions for that sector. Thus, a constraint added by a reflex on a *space-variant action feature* has the following format: $cons(j, f_{sv}, c, \tau)$ (e.g. $cons(2, v_{max}, 1 \text{ ms}^{-1}, 100 \text{ ms})$).

A set of conditions must be met when accepting a new constraint. If a new constraint reduces the possible set of actions (e.g. if the feature is v_{max} with a current value of 1 ms^{-1} and the new constraint intends to reduce the value to 0.5 ms^{-1}), then it is immediately accepted. If the new constraint validity is greater than the current one, then the constraint validity is updated with the newer value.

The constraint validity can be used for three purposes: to reduce the system sensitivity to noisy sensors, to specify Fixed Action Patterns, and to create a myopic local environment representation in terms of *space-variant action features*. The representation is myopic due to the absence of self-localisation mechanisms, which, in general, hinders spatial transformations, specially translations. However, if heading information exists, the *space-variant action feature sub-space* can be geo-referenced. In such case, if the robot is pointing north and a constraint is set to sector $j = 0$, while still valid, that constraint will always refer to the sector pointing north independently of robot's subsequent headings.

Space-Invariant Action Feature Sub-Space A second action feature sub-space, the space-invariant action feature sub-space, allows reflexes to constrain certain dynamics of the robot, such as angular velocities, independently of any spatial relationship. Setting a constraint in this sub-space is in everything similar to the previous case, except that a feature $f_{si} \in F_{si}$ is considered, and any spatial relationship discarded. Thus, a constraint is defined by $cons(f_{si}, c, \tau)$. Notice that $F_{sv} \cap F_{si} = \emptyset$.

2.2 Reflex Pools

Two loops can be found in the SK architecture. A set of reflexes perceives the world, acts upon the *action feature space* (i.e. produce a set of constraints), an action is selected which will change the world, and in turn the world affects robot's perception. This is the first loop. The second loop is internal, where reflexes are able to sense actions produced by other reflexes, a sort of *internal stigmergy*. There is also some resemblance with the concept of *black board*.

To implement the second loop, reflexes are split into several pools, which are iterated in sequence. The system designer can allocate reflexes to pools as required. In general, reflexes of pool n will support their actions on information set by reflexes of pools $m < n$, described in terms of constraints.

To better understand this concept, let us assume the existence of two reflex pools. So, reflexes of the first pool act according to the current state of the internal (i.e. *action feature space*) and external (i.e. sensory information) environments. Reflexes of the second pool act in the same manner; however, since the *action feature space* already considers constraints set by reflexes of the first pool, reflexes of the second one can act accordingly.

2.3 Descendent Pathways

Upper-layers are allowed to modulate the SK, via two descending pathways, one inhibitory and other excitatory. Via the inhibitory pathway, upper-layers can constrain the *action feature space* as if they were reflexes. The SK requests those behaviours for their contribution (i.e. constraints), before any other reflex pool. To increase plasticity, the inhibitory pathway also allows upper-layers to suppress reflexes' output; otherwise, some high-level behaviours would be impossible to achieve (e.g. obstacle avoidance would impair a docking behaviour).

In opposition to the inhibitory pathway, the excitatory one is exclusive; hence, in order to guarantee that only one behaviour can send a message to the SK through this channel, a coordination mechanism must be provided. The excitatory modulation signal allows to select, configure, and request the action selection mechanism for a certain motor action pattern (e.g. *desired heading*, and *desired linear velocity*). The SK architecture could include a coordination node allowing several behaviours to send excitatory signals, which would require to commit to a coordination philosophy. Since the goal of the SK architecture is confined to provide safe navigation, such decision is left to upper-layers.

2.4 Action Selection

The SK can aggregate as many *action selection* mechanisms as desired; still, only one can be active at a time, which is selected and configured via the descendent pathway. Constrained by the available action features, and modulated by the excitatory signal, the *action selection* module selects the best action according to a given criteria.

The *motor actions producer* module is responsible for converting the best action into robot's actuators outputs, and to cooperate with the *action selection* mechanism so to guarantee that a certain action is both feasible in terms of actuators, and respects both *action feature sub-spaces* constraints. For instance, the *space-variant sub-space* may allow the robot to move in sectors on the right of the robot, whereas the *space-invariant* one inhibits positive angular velocities (i.e. turning right). This apparent contradiction is only relevant in some locomotion methods, such as differential ones, and not in others, such as those present in omni-directional robots.

We define the concept of *main axis of motion*, which may not coincide with the robot's front. This concept allows to handle different types of locomotion within the same framework. The *main axis of motion* is displaced by an angle \hat{a} from robot's front, and represents the side of the robot that should match the *desired heading* (provided in the excitatory signal). In other words, this feature allows to set the *desired posture* of the robot.

3 A Survival Kit Instance

In order to demonstrate the SK architecture, an instance for a simulated indoor differential robot with a rectangular shape is herein presented. The robot carries

a 5 m full-sonar ring, composed of 32 evenly separated elements, a set of bumpers, and a compass.

3.1 Action Feature Space

The *space-variant action feature* set is $F_{sv} = \{v_{max}, d_{max}, enable\}$, which represents for a given sector, the maximum linear velocity allowed, the maximum distance free of obstacles, and a flag indicating if movement in that sector is allowed at all, respectively. When a feature is constrained, a limit value will be established. Constraints associated to action features v_{max} and d_{max} are real, whereas those associated to $enable$ are boolean. This action feature sub-space is *geo-referenced*. Notice that constraints in the following text are described in *ego-referenced* coordinates; an internal process is responsible for making them *geo-referenced*.

The *space-invariant action feature* set is $F_{si} = \{\omega^+, \omega^-, stop\}$. These features, when constrained, disable positive angular velocities, negative angular velocities, and the possibility of stopping the robot, respectively. Thus, in this sub-space, constraints are boolean.

3.2 Reflexes

Five reflexes have been implemented: *range-based reflex*, *touch reflex*, *shape constraints reflex*, *linear velocity constraint reflex*, and *dynamic constraints reflex*. The first three are contained in the first reflex pool, the fourth reflex in the second reflex pool, whereas the last reflex is located in the third pool.

Range-Based Reflex. In the experiments performed for this work, the robot uses a full-sonar-ring of 32 elements, which are displaced by $\alpha = \frac{2\pi}{32}$ radians. An obstacle i has the following format, (β_i, r_i) , where β_i is the angle between the front of the robot and the obstacle's direction, and r_i is the distance to the obstacle.

In the following text, the term obstacle will be used to something that is perceived by a sonar, and not necessarily to a complete object in the environment (see figure 2.a); that is to say that there is an obstacle per each sonar that detects something. This approach avoids the computational burden of creating elaborate representations of obstacle shapes as they are in the environment. The d_{max} feature of each sector containing an obstacle i , is constrained by the range data r_i .

It is necessary to enlarge the obstacle so a sector not containing it is in fact navigable; i.e. obstacles have to be considered in the configuration space (c-space). This work simplifies this problem by exploiting the fact that obstacles' angular size is equal to the sonar's field of view (i.e. it is fixed). Hence, taking into account the ratio between obstacle's diameter, roughly defined as $d_i = r_i \cdot \sin \alpha$, and robot's diameter w , it is possible to determine, in the c-space, all sectors

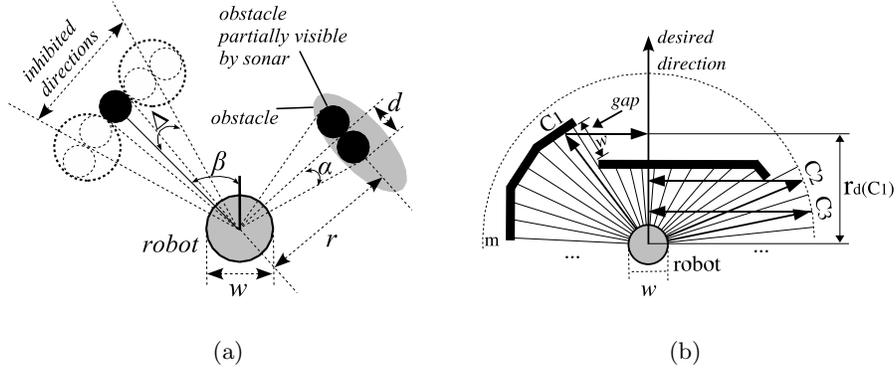


Fig. 2. The c-space transformation in a). The gaps mechanism in b), where obstacles are in c-space and a *corridor* is represented by C_i . On the sake of simplicity, only corridor C_1 has its range projected over the desired direction. It will be the chosen one because $r_d(C_1)$ is the largest of all $r_d(C_i)$.

affected by obstacle i ; namely, $j_{obs_i} \in [\Omega(\Delta_i - \beta_i), \Omega(\Delta_i + \beta_i)]$, where $\Omega(\cdot)$ returns the sector of a given angle, and $\Delta_i = \frac{1}{2} \cdot \alpha \cdot \left(\frac{w}{r_i \cdot \sin \alpha} + 1 \right)$.

Further, as previously referred, the d_{max} feature of each sector affected by the enlarged version of the obstacle must be constrained. This is performed by adding, for each sector j_{obs_i} , a constraint $cons(j_{obs_i}, d_{max}, r_i, 100 \text{ ms})$. The constraint validity has been set empirically. The process is repeated for all detected obstacles.

Touch Reflex. This reflex intends to handle situations where the robot collides with obstacles, which by their nature, e.g. small height, are not detectable by the sonar ring. When a bumper is triggered, this reflex reports the obstacle by adding a constraint $cons(j_{touch}, d_{max}, 0, 4000 \text{ ms})$ per each $j_{touch} \in [\Omega(-\frac{\pi}{2}), \Omega(\frac{\pi}{2})]$. To avoid the robot to rotate as it moves away from the obstacle, constraints $cons(\omega^+, false, 2000 \text{ ms})$ and $cons(\omega^-, false, 2000 \text{ ms})$ are also set. Finally, to guarantee that the robot actually moves, i.e. it does not choose to stop, the constraint $cons(stop, false, 2000 \text{ ms})$ is added. The way the robot moves away from the obstacle will be a function of the constraints and excitatory modulation signal. Briefly, after the collision, the robot moves straight backwards for two seconds. Afterwards, it turns away from the detected obstacle bearing towards best direction; progressively the robot gains velocity. Meanwhile the d_{max} constraint validity expires and the robot moves as that obstacle is not there anymore. Thus, this reflex triggers a Fixed Action Pattern described in terms of validity constraints that helps the robot to contour an obstacle invisible to the range sensors.

Shape Constraints Reflex. To consider robot shapes different from the circular one, it is regularly necessary to take into account that, when the robot turns, even around its centre, it may hit an obstacle. Instead of considering robot’s shape explicitly in a model-based decision making process, a reflexive method is proposed. Let us assume a situation where a rectangular robot, which due to the current *desired heading* and absence of obstacles on the right side of the robot, decides to turn right. However, a small obstacle hits the back left side of the robot as soon as it turns. To avoid that situation, every time an obstacle is detected in that area, a constraint $cons(\omega^+, false, 500\text{ ms})$ is added.

Defining constraints to other similar situations endows the controller with the ability to negotiate cluttered environments with non-circular robots, without explicitly modelling them. A careful choice of sensors localisation may discard the need of a full-sonar-ring set for this reflex.

Linear Velocity Constraint Reflex. The maximum linear velocity allowed for each sector j , considering a linear trajectory and uniformly accelerated movement, is $v_{allowed}(j) = \sqrt{2 \cdot a_{max} \cdot \max(0, r_j - d_{safety})}$, where a_{max} is the maximum deceleration capability of the robot, r_j is the minimum linear distance free of obstacles in the respective sector, d_{safety} is the minimum distance the robot is allowed to be from an obstacle, and max guarantees that $r_j - d_{safety}$ is never negative. Thus, the maximum velocity allowed is such that guarantees the robot does not hit any obstacle when decelerating as much as it can. This acceleration can be set for other purposes in addition to dynamical constraints; for instance, to guarantee that robot’s payload does not suffer accelerations and vibrations greater than certain values. Then, a constraint $cons(j, v_{max}, v_{min}, 100\text{ ms})$ is added per each sector, where v_{min} is the minimum velocity of all $v_{allowed}$ calculated in the sectors between the *main axis of motion* and j .

Since this reflex is in the second pool, it can measure the free distance to obstacles by perceiving the d_{max} action feature of the corresponding sector (i.e. to make $r_j = d_{max_j}$), which has been set by reflexes of the first pool, and already considers the c -space. Consequently, this reflex takes into account information set by all previous reflexes. The constraint validity has been set empirically.

Dynamic Constraints Reflex. Considering the robot’s dynamical constraints, this reflex disables those sectors that would require unfeasible actions. In particular, a constraint $cons(j_{dyn}, enable, false, 1\text{ iteration})$ is added per each sector j_{dyn} ; j_{dyn} is a sector whose v_{max} is smaller than the robot’s minimum velocity achievable until the next iteration (i.e. taking into consideration a_{max}).

3.3 Action Selection Mechanism

The implementation of the action selection mechanism is based on the environmental feature *gap*. It represents a discontinuity greater than the width, w , of the robot in the d_{max} feature between two adjacent sectors. From the two sectors involved in the discontinuity, the one with greater d_{max} is considered to be a

corridor. Since a *corridor* is defined in the c -space (by means of d_{max}), the robot is kept safe of collisions as it moves in it (see figure 2.b). Every sector in which the d_{max} feature is greater or equal to m (a value provided in the excitatory signal) is also considered to be a *corridor*. Reducing m allows the robot to move towards obstacles and consequently approaching a goal set between itself and the obstacle. Finally, $d_{max_i} = \min(m, d_{max_i})$.

The best action of the current iteration n is composed of a *selected corridor* and a *selected linear velocity*. If the maximum linear velocity allowed in the *selected corridor* is smaller or equal than the desired linear velocity, provided in the excitatory signal, then it is selected; otherwise, the desired linear velocity is the one selected. After discarding all *corridors* that have been either inhibited or rejected by the *motor actions producer* (see below), the *selected corridor*, s_c , is chosen by maximising an objective function:

$$s_c[n] = \max_i (\lambda_1 \cdot r_d(\text{corridor}_i) + \lambda_2[n] \cdot \text{dist}_\angle(\text{corridor}_i, s_c[n-1]))$$

where, the first term (r_d) provides goal-oriented behaviour, the second term (dist_\angle) allows to *commit* to the previously selected *corridor* in order to reduce oscillatory behaviour (similar to the definition provided in [11]), and weights λ_1 and $\lambda_2[n]$ enable a proper integration of both components, all normalised between $[0, 1]$.

$r_d(\text{corridor}_i)$ refers to the d_{max} value of corridor_i projected on the desired direction axis (see figure 2.b). dist_\angle grows with the angular distance between $\text{geo}_\angle(\text{corridor}_i)$ and $\text{geo}_\angle_{[n-1]}(s_c[n-1])$, where $\text{geo}_\angle(\cdot)$ transforms the bisector angle of a given sector to geo-referenced coordinates, whereas $\text{geo}_\angle_{[n-1]}$ does the same for iteration $n-1$. This mechanism is simple to tune due to the little number of parameters, which only exist to reduce possible oscillations.

In order to remove heading static errors, λ_2 is made dynamic. Briefly, if the robot's heading is nearby the desired heading, and the selected action does not change significantly between consecutive iterations (i.e. $\{(\text{heading error} < \frac{\pi}{8}) \wedge (\angle(\text{geo}_\angle(s_c[n]), \text{geo}_\angle_{[n-1]}(s_c[n-1])) < \phi)\}$, where \angle returns the angular distance between two given angles), then λ_2 is decreased at rate τ_{λ_2} ; otherwise, λ_2 is set to its maximum value, α . As a result, in the presence of heading static error the dist_\angle component is reduced and the robot becomes more goal-oriented. If instead a sudden change of heading is required (e.g. due to the presence of an obstacle) the former weights trade-off is restored.

If no action is possible, then the *action selection* mechanism switches the *main axis of motion* to other possible value and tries again. Since the robot is differential, it is only reasonable to assume that the *main axis of motion* is either 0 or π ; i.e., that the robot will either try to move forward or backwards, respectively. This is in fact what happens in the *Touch Reflex*, where no motion is possible if moving forward, which forces to try moving backwards.

As aforementioned, the robot's locomotion method is differential, i.e. an action is composed of a *linear velocity* and an *angular velocity*; so, it is non-holonomic. The linear velocity is set by the *action selection* mechanism as previously described, and the angular velocity is proportional to the angle between

the *selected corridor* and the robot’s front. This process is implemented within the *motor action producer*, which also confirms the feasibility of an action, taking into consideration the differential locomotion method.

4 Experimental Results

The experiments herein presented were carried out in the Player/Stage simulator[5]³. Acceleration ramps have been implemented between the simulator and the control system, to emulate the robot’s dynamical constraints (i.e. a_{max}).

Figure 3 illustrates two experiments of the SK instance presented in the previous section. In both experiments, the SK is modulated to move towards north (top of the figure) at a speed of 2ms^{-1} . λ_1 , α , a_{max} , m , d_{safety} , and τ_{λ_2} have been set to 1.0, 0.75, 0.5ms^{-2} , 5 m, 0.5 m, and 1 s respectively. The *space-variant action feature sub-space* is split into 128 sectors. The *gap* feature allows detection of the dead-end in run 1, which is not present in the second run, allowing the robot to follow the shorter path. In run 1 the touch reflex is triggered forcing the robot to move backwards (situation A). Notice that, the robot moves backwards while rotating towards the best direction (i.e. left), which demonstrates the cooperative nature of the method. In the beginning of the second run, the robot can not move towards north due to the presence of an obstacle, which triggers the shape reflex. In some situations, such as situations B and C, the *concave* nature of the environment is responsible for the emergence of *fake gaps*. In situation B, the robot temporarily turns right towards the *corridor* containing the *fake gap*, whereas in situation C, the robot is insensitive to the *fake gap* due to the effect of the myopic local environment representation, which allows the robot passing by the situation before it realises the existence of the *fake gap*.

5 Discussion

Although the SK instance herein presented does not explicitly accommodate arc trajectories, it guarantees collision free navigation. Nevertheless, the main advantage of this approach is its simplicity as it discards complex geometric reasoning while keeping robust navigation; thus, the requirements for disposable robots have been attained. In fact, it is questionable the need of handling, in a refined way, kinematics and dynamical constraints for simple and small robots. Still, if arc trajectories have to be considered in a more sophisticated way, then a w_{min} action feature could be added. Such feature would be set to the widest arc trajectory, represented by the tuple (v_{max}, w_{min}) , that would lead the robot to the sector in question without crossing by any obstacle. Then, angular velocity dynamical constraints could be considered in the same way linear velocities already are. In the limit of the aforementioned extension, only one arc would have to be computed by *corridor*, whereas in the *Dynamic Window Approach* [4] and

³ <http://playerstage.sourceforge.net/>

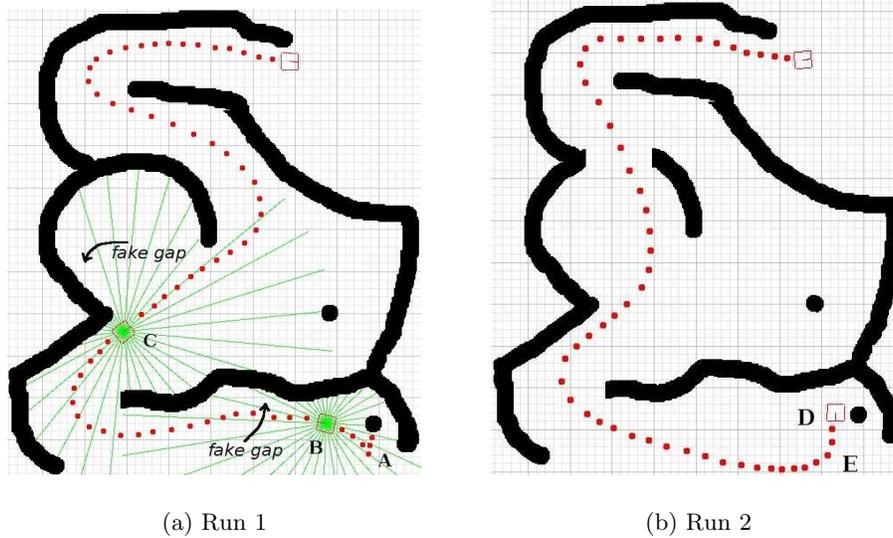


Fig. 3. Set of experiments using Player/Stage simulator.

Curvature Velocity Method [9] more than a single arc has to be considered per *obstacle*. Hence, even with this improvement, the SK architecture would remain simpler. In addition, the SK instance herein proposed does not suffer from local minima as *Potential Fields* based methods (e.g. [1]) in general do, and it deals with dynamical constraints in opposition to the *Nearness Diagram* method [7].

Since r_d already embodies information on the distance to obstacles, heading error, and speed, the objective function is simpler and more predictable than the ones used in the *Dynamic Window Approach*, *VFH+* [11], and *Curvature Velocity Method*, where all those features have to be considered explicitly.

Previous work used the *gap* feature, but none in the same way as introduced in this work. In the *Gap Navigation Trees* [10], the *gap* environmental feature is used as a mean of producing a topological map; however, a reactive obstacle avoidance method is still required. The SK architecture merges, to some extent, optimised navigation and reactive obstacle avoidance instead. The concept of *gap* was also used in the *Nearness Diagram* method, but not as a direct target of motion; instead, it is exploited differently in five separable situations, which requires additional computational load.

6 Conclusions

First experimental results in simulation suggest that the SK architecture, via cooperation by negation (i.e. constraints) and modulatory signalling, is a good

option for safe navigation, while locally optimising robot's trajectory; in other words, it merges the benefits of both cooperative and competitive-based Behaviour-Based architectures. The *gap* feature is a promising approach to optimise trajectories and avoid dead ends based on immediate ranging data; moreover, it allows the application of a simpler objective function than those found in previous work. Dynamical constraints are also considered in a simplistic but robust fashion. Finally, the resulting behaviour is predictable, local-minima and oscillations free, and it does not require upper-layers supervision, just modulation. The low-computational load that this method requires is of special interest for disposable robots, which can only afford simple computational units, such as micro-controllers. In fact, it is the trade-off between performance and low computational load that makes this approach original and interesting.

7 Acknowledgements

The authors wish to thank Pedro Mariano and Ana Belchior by their valuable comments.

References

1. Arkin, R.C.: Motor schema-based mobile robot navigation. *International Journal of Robotics Research* **8**(4) (1989) 92–112
2. Brooks, R.: A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* **2**(1) (1986) 14–23
3. Correia, L., Garçon, A. S.: Behavior Based Architecture with Distributed Selection. In *NATO Advanced Study Institute – The Biology and Technology of Intelligent Autonomous Systems* **144** (1995) 377–389
4. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine* **4**(1) (1997) 23–33
5. Girkey, B.P., Vaughan, R.T., Howard, A.: The Player/Stage Project: tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics (ICAR)* (July 2003)
6. Maes, P.: How to do the right thing. *Connection Science Journal* **1**(3) (1989) (291–323)
7. Minguez, J., Montano, L.: Nearness diagram navigation (ND): collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation* (2004) **20**(1) (2004) 45–59
8. Rosenblatt, J.K.: DAMN: a distributed architecture for mobile navigation. In *Proceedings of the AAAI Spring Symp. on Lessons Learned from Implemented Software Architectures for Physical Agents* (1995)
9. Simmons, R.: The curvature velocity method for local obstacle avoidance. In *Proceedings of the IEEE International Conference on Robotics and Automation* (1996)
10. Tovar, B., Guilamo, L., LaValle, S.M.: Gap navigation trees: a minimal representation for visibility-based tasks. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics* (2004) 11–26
11. Ulrich, I., Borenstein, J.: VFH+: reliable obstacle avoidance for fast mobile robots. In *Proceedings of the International Conference on Robotics & Automation* (1998)