CAT Finland: Executing Primitive Tasks in Parallel

Jukka Riekki, Jussi Pajala, Antti Tikanmäki & Juha Röning
Dept. of Electrical Engineering and Infotech Oulu
University of Oulu
FIN-90570 Oulu
{jpr,jussip,sunday,jjr}@ee.oulu.fi

Abstract. We present a novel representation for agent actions. An action map specifies the preferences for all the possible different actions. The key advantage of this representation is that it facilitates executing in parallel primitive tasks. We have utilized the action map representation in playing soccer. We describe here the system that controlled our players in the second RoboCup competition.

1 Introduction

A mobile agent operating in a dynamic environment needs the capability to react to several objects in the environment in parallel. Some reactions progress the task at hand, while some help to avoid obstacles and other mobile agents. We call these tasks of reaching and avoiding objects primitive tasks and the corresponding goals primitive goals.

In this work, we studied a situated approach, where the actions of the agent are calculated directly based on the information collected by the sensors from the environment [Brooks, 1991; Agre and Chapman, 1990; Kaelbling and Rosenschein, 1990; Arkin, 1990]. For each primitive task, there is a module producing commands that progress the task. Executing tasks in parallel by combining these commands is difficult, because the commands do not contain enough information to allow selecting a compromise that would satisfy several primitive goals.

Parallel task execution is possible to some extent in the architecture suggested by Rosenblatt and Payton [1989]. In this architecture, the modules produce votes for different commands. The votes are then summed and the command with the maximum sum is selected. However, this approach has limitations in a dynamic environment. As the heading and speed commands are processed separately, it is difficult to guarantee that the agent will reach the right location at the right time.

In this paper, we present a method for executing primitive tasks in parallel. The commands produced by independent modules are combined by a simple operation. This method is based on a novel representation of agent actions – the action map representation. The method is applied to playing soccer.

2 Action Maps

An action map specifies for each possible action how preferable the action is from the perspective of a task. The preferences are shown by assigning a weight to each action. A separate action map is calculated for each primitive task of reaching or avoiding an object. More complex tasks are performed by combining these primitive action maps.

The action maps controlling the way the agent reaches and avoids objects are called velocity maps, as they contain weights for different agent velocities. The weights make up a two-dimensional surface in a polar coordinate system (direction, speed). A weight for a velocity is calculated based on a global optimality criterion, the time to collision. The shorter the time needed to reach an object, the heavier the weight for that action. For actions that do not result in a collision, the weights are calculated by propagating, i.e., by adjusting the weights towards the weights of the neighboring actions on the map.

M. Asada and H. Kitano (Eds.): RoboCup-98, LNAI 1604, pp. 396-401, 1999.

To preserve the weights of the actions that do reach the object, the weights are adjusted only upwards.

The resulting map contains only positive weights. A map of this kind is called a Goto map, because the performance of the action with the currently heaviest weight on such a map causes the agent to reach the corresponding object. An example of a Goto map is shown in Fig. 1.

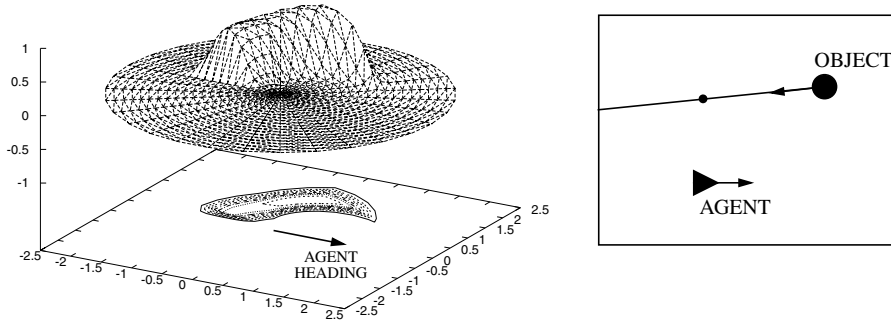

Fig. 1. Left: A Goto map. **Right:** The situation. The agent reaches the object at the location marked with the small dot, if it executes the action with the heaviest weight on the Goto map.

An action map containing negative weights is called an Avoid map. It is calculated by negating the corresponding Goto map. As only positive weights cause actions to be performed, an Avoid map does not alone trigger any motion, but only prevents some actions. In other words, an Avoid map prevents collision with an obstacle when a target is being reached with the help of a Goto map. Fig. 2 shows an Avoid map.

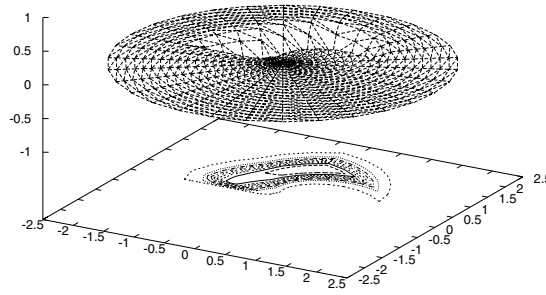

Fig. 2. An Avoid map corresponding to the Goto map shown in Fig. 1.

Tasks are executed in parallel by combining primitive Goto and Avoid maps into composite maps by the Maximum of Absolute Values (*MAV)* method. In this method, the weight with the maximum absolute value is selected for each action. In other words, the shortest time it takes to reach an object is selected for each action and the corresponding weight is stored on the composite map. Thus, the sign of a weight on a composite map specifies whether an obstacle or a target would be reached first if a certain action were performed. The absolute value of the weight specifies the time to collision with the object that would be reached first. The global optimality criterion guarantees that this method provides correct results. Fig. 3 illustrates the MAV method for compiling action maps.

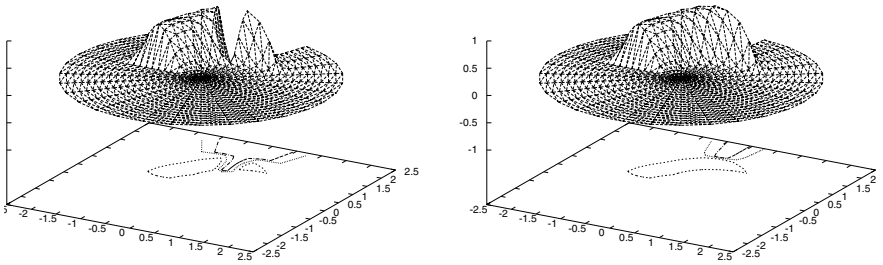

Fig. 3. A Goto map for a moving target and an Avoid map for a stationary obstacle compiled with the *MAV* method. **Left:** The obstacle is in front of the trajectory of the target, and actions in that direction are therefore forbidden. **Right:** The obstacle is behind the trajectory, and no actions are hence forbidden.

Sometimes we need to modify maps before combining them. For example, we might want to consider only actions that reach the target fast enough. Or, when avoiding an obstacle, we might want to allow those actions that cause a collision only after a considerable length of time. Modifications of this kind can be produced by filtering. Fig. 4 shows the result of filtering actions that require too long time to reach a target. Filtering can also be performed by utilizing a filter mask containing a zero for each action to be filtered and one for the rest of the actions.

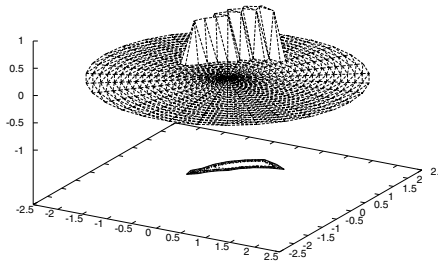

Fig. 4. Effect of filtering. The actions requiring more than 1.1 seconds to reach a target have been filtered from the Goto map shown in Fig. 1.

3 Soccer

Action maps have been utilized in playing soccer. As the control system controls the movements of both the player and the ball, the system produces two types of action maps. A velocity map specifies preferences for the different actions for reaching an object. An impulse map presents weights for all the possible different impulses (direction, magnitude) the player can give to the ball. It specifies the preferences for the different actions for kicking the ball to reach an object. Table 1 presents the primitive maps.

The opponent and team-mate maps are calculated for all opponents and team-mates whose locations are known. The GotoLoc map is utilized when a player is to kick a free kick, a goal kick, a corner kick, or a side kick. The GotoDefloc map controls a player to its default location. The KickNear and KickFar maps contain heavy weights for impulses that cause the ball to move nearer the opponents goal. The KickNear map favors small impulses, whereas the KickFar map favors big impulses.

Table 1: The primitive maps.

MAP TYPE	PRIMITIVE VELOCITY MAPS	PRIMITIVE IMPULSE MAPS
Goto	GotoBall, GotoTeam, GotoOpp, GotoLoc, GotoDefloc	KicktoGoal, KicktoTeam, KicktoOpp, KickNear, KickFar
Avoid	AvoidBall, AvoidTeam, AvoidOpp, AvoidLoc, AvoidDefloc	DontKicktoGoal, DontKicktoTeam, DontKicktoOpp, DontKickNear, DontKickFar

The primitive maps are filtered, as pointed out in Chapter 2: when avoiding opponents, the actions that require a considerable amount of time to reach the opponent are filtered from the Avoid maps. The ForbiddenSector mask is an example of a filter mask. This mask contains a zero for each impulse that would cause the ball to collide with the player kicking the ball. Fig. 5 shows an example of a forbidden sector mask.

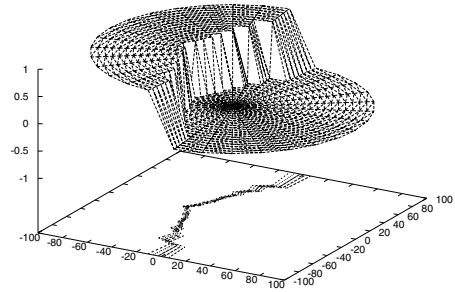

Fig. 5. A forbidden sector mask for a situation in which the ball is behind the agent and moving towards the agent. To prevent a collision, the agent should kick the ball backwards hard enough.

The composite maps are listed in Table 2. The MovetoSpecloc map controls the agent towards a good location (special location) when a goal kick, a free kick, a side kick, or a corner kick is to be performed. In this map, the AvoidBall map prevents only actions that would result in a collision with the ball in less than 1.5 seconds. Only one teammate and opponent map of a type is listed for each composite map, although all teammate and opponent maps are utilized. Each of the composite impulse maps is filtered with the ForbiddenSector mask.

Table 2: The composite maps.

MovetoSpecloc = MAV(GotoLoc, AvoidOpp, AvoidTeam, filter(AvoidBall, 1.5))
MakeGoal = filter(MAV(KicktoGoal, DontKicktoOpp, DontKicktoTeam)), ForbiddenSector)
Pass = filter(MAV(KicktoTeam, DontKicktoOpp), ForbiddenSector)
Dribble = filter(MAV(KicktoNear, DontKicktoTeam, DontKicktoOpp), ForbiddenSector)
CatchBall = MAV(GotoBall, AvoidOpp, AvoidTeam)
MovetoDefloc = MAV(GotoDefloc, AvoidOpp, AvoidTeam)

A task is performed by sending the action with the heaviest weight in the appropriate composite map to the actuators. The task to be executed at each moment is selected by an arbiter, which goes through the tasks in the following order: SpecialSituation, LookAround, MakeGoal, Pass, Dribble, CatchBall, MovetoDefloc, and TurntoBall. The arbiter selects the first task whose valid conditions are fulfilled. When a task has been selected, it remains active as long as its valid conditions are fulfilled. Among the tasks listed above, LookAround and TurntoBall are executed without utilizing action

maps. The SpecialSituation task expands into a prioritized task set similar to the list above. This task set contains the MovetoSpecloc task listed in Table 2.

These task sets and the valid conditions of the tasks produce the following behavior: As long as a special situation (a free kick, goal kick, etc.) is observed, the agent performs the SpecialSituation task. Otherwise, if the location of the ball or the location of the agent is not certain the agent looks around until the certainty increases to an acceptable level. Otherwise, if the ball is near enough, the agent kicks it. It favors scoring a goal. If the weight of the best action to score a goal is too low, it considers passing the ball to a team-mate. If this cannot be done, either, the agent dribbles the ball. If the ball is not near enough to be kicked, but no team-mate is nearer the ball, the agent tries to catch the ball. Otherwise, if the agent is too far from its default location, it runs towards it. Otherwise, the agent turns towards the ball.

4 Discussion

When reactions to objects in the environment are calculated as action maps, the resulting system is situated. All reactions are grounded to the situation around the agent. Furthermore, action maps allow primitive tasks to be executed in parallel. Specifically, they allow tasks to be assessed separately and combined by a simple operation. This advantage facilitates incremental development.

The difference between this and the related work is in the amount of information encoded in a command. A larger amount of information encoded in action maps allows more versatile combination operations than the command representations in the related work. Specifically, by combining action maps, it is possible to meet primitive goals in parallel in a dynamic environment. Furthermore, action maps allow time provisions. Such rules as “Go after the ball if you can reach it in less than 2 seconds”, or “Try to score a goal if the ball would reach the goal in less than 3 seconds” are easily implemented by operations on action maps.

We have tested action maps in the Soccer Server’s simulated world at the RoboCup’98 in Paris. The experiments show that the action maps are a potential alternative for mobile agents operating in dynamic environments. Even when playing against the finalist, AT Humboldt, the game was quite even for long periods. This was due to our players’ skill of moving to a good location and skill of selecting were to kick. However, our team did not win AT Humboldt, as our players were not good enough in dribbling, had only some simple strategies, and did not always behave in a stable way. Our goalkeeper also made some bad misses.

Soccer as an application has had a major impact on our research. The Soccer Server has facilitated performing a large amount of experiments in a highly dynamic environment. Soccer is also an interesting application, because it shows how difficult it is to decide on paper which situations are important in an application and how the agent should operate in those situations. For example, it seems to be important for a soccer player to avoid other players. Actually, the worst thing to do is to avoid an opponent approaching you. Furthermore, the skill of avoiding opponents has no noticeable correlation with the results. Due to modifications in the Soccer Server, we even run some experiments in which the opponent could not see our team. This was not obvious when observing the game. The opposing team played quite an effective game when they were not aware of our players. Further, another important skill in soccer seems to be the ability to stay on the field. However, our team plays a better game when they do not consider the boundary lines at all. As all the other players and the ball are on the field, reactions to them tend to keep also the reacting player on the field.

Maybe the most important lesson we have learnt for soccer is the improved insight about building situated agents. We have learnt that it is not necessary to be able to find in the problem space the optimal path (or even any path at all) to the goal state from every possible state. Instead, it suffices to find a feasible action that decreases the distance to the goal state. Action maps are one example: if the optimal action is prevented, an action guiding the agent closer to the goal location is selected. Another example is the behavior of our player when it is otherwise in a good location to score a goal, but the ball would bounce from the player itself if kicked towards the goal. One solution would be to calculate a sequence of kicks that would first move the ball around the player and then to the goal. Our player selects some other action in such a case. For example, if the player chooses to dribble the ball instead, it will probably be possible to score a goal a few moments later.

In other words, in a complex and dynamic environment the behavior satisfying the goals of the agent can emerge from a carefully selected set of primitive tasks. In addition to being situated, the resulting system is also considerably simpler compared to one planning optimal actions. Our players never plan ahead, they just calculate the next (turn, dash) action pair. An example of an emerging behavior is an attack performed by our team: there is no mechanism controlling the players towards the opponent's goal just because the team is attacking. Instead, the players favor the direction of the opponent's goal when kicking the ball. The location of the ball, in turn, changes the default locations of the players. As a result, when our team has the ball, they kick it towards the opponent's goal and move themselves to the same direction. In other words, they attack.

5 Conclusions

In this paper, we suggested a novel representation for agent actions. An action map specifies preferences for all the possible different actions. The central advantage of this representation is that it facilitates executing in parallel primitive tasks, i.e., tasks of reaching and avoiding objects. We have applied the action map representation to playing soccer and carried out experiments in the second RoboCup competition.

Acknowledgments

The research presented in this paper was supported by the Tauno Tönning Foundation.

References

[Agre and Chapman, 1990] Agre PE & Chapman D (1990) What are plans for? In: Maes (ed) Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back. MIT Press, Cambridge, MA, p 105-122.

[Arkin, 1990] Arkin RC (1990) Integrating behavioral, perceptual, and world knowledge in reactive navigation. In: Maes (ed) Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back. MIT Press, Cambridge, MA, p 17-34.

[Brooks, 1991] Brooks RA (1991) Intelligence without representation. Artificial Intelligence (47): 139-159.

[Kaelbling and Rosenschein] Kaelbling LP & Rosenschein SJ (1990) Action and planning in embedded agents. In: Maes (ed) Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back. MIT Press, Cambridge, MA, p 35-48.

[Rosenblatt and Payton, 1989] Rosenblatt JK & Payton DW (1989) A fine-grained alternative to the subsumption architecture for mobile robot control. Proc IEEE/INNS International Joint Conference on Neural Networks, Washington DC, 2: 317-324.