Behavior Based Robotics Using Hybrid Automata

Magnus Egerstedt*

Division of Optimization and Systems Theory
Royal Institute of Technology
SE - 100 44 Stockholm, Sweden
magnuse@math.kth.se

Abstract. In this article, we show how a behavior based control system for autonomous robots can be modeled as a hybrid automaton, where each node corresponds to a distinct robot behavior. This type of construction gives rise to chattering executions, but we show how regularized automata suggest a solution to this problem. We also discuss some design and implementation issues.

1 Introduction

For mobile, autonomous robots the ability to function in, and interact with a dynamic, changing environment is of key importance. A successful way of structuring the control system in order to deal with this problem is within a *behavior based* control architecture [3]. The main idea is to identify different controllers, responses to sensory inputs, with desired robot behaviors. A behavior could, for instance, be obstacle avoidance in which sonar information about a close obstacle should result in a movement away from that obstacle. This way of structuring the control system into separate behaviors, dedicated to performing certain tasks such as avoid obstacles or traverse doors, has turned out to be a successful design. It has the major advantage that it makes the system modular, which both simplifies the design process as well as offers a possibility to add new behaviors to the system without causing any major increase in complexity.

The suggested outputs from the different, concurrently active behaviors are fused together according to some action coordination rule, and this makes it easy to stress such questions as safety explicitly, since, for example, an avoidance behavior can just be given higher priority than a reach target behavior.

However, within this framework, a number of design issues still need to be addressed. Those range from questions concerning the design of the individual behaviors to action coordination issues [5]. For instance, given a reactive obstacle avoidance behavior, modeled as a repulsive field surrounding the obstacle, how should an approach target behavior be designed so that it takes advantage of

* This work was sponsored in part by the Swedish Foundation for Strategic Research through its Centre for Autonomous Systems at KTH.

N. Lynch and B. Krogh (Eds.): HSCC 2000, LNCS 1790, pp. 103–116, 2000.

the fact that it is going to run in parallel with an obstacle avoidance behavior? Furthermore, how should these behaviors be combined?

What will be investigated in this article is how a behavior based system can be modeled as a *hybrid automaton* with each of the discrete nodes corresponding to a distinct behavior. If the system where to be described by such an automaton it would hopefully help us understand and explain some of the so called *emergent* phenomena that complex robotics systems can give rise to. We will furthermore see that questions concerning safety and optimality can be addressed nicely within this framework.

The outline of the article is as follows: First, in Section 2, we discuss some of the properties of a behavior based robotics system, and we show how this can be modeled as a hybrid automaton. Some regularization techniques are then exploited in order to get rid of potential chattering in the automaton. In the next section, some control design issues are discussed, and we describe a heuristic method for constructing behaviors that are safe at the same time as they are close to optimal with respect to a given performance evaluation functional. We conclude, in Section 4, with a brief discussion about a proposed, systematic strategy for implementing the hybrid automata.

2 Behavior Based Robotics

As already mentioned, for autonomous robots operating in a partially unknown, dynamic environment a successful way of structuring the controllers is within a behavior based framework [3],[10]. Different robot behaviors are identified, e.g. obstacle avoidance or reach target, and their functionality is defined by a tight mapping from sensory data to a desired action. Typically, in a so called *reactive* behavior based system, no representation of the world is contained in this mapping, while a *deliberative* system exploits planning or world models in the control loops.

The desired output actions are then normally fused together by an arbitration mechanism, as seen in Figure 1, where a wide-spread solution to the action fusion problem, used for example in the schema theoretical paradigm, is to represent the goals, targets and obstacles by weighted attractive or repulsive potential fields, resulting in weighted, desired orientation vectors. The action coordination is then simply done using vector summation. This way of letting behaviors be active simultaneously is desirable in many situations. For instance, while approaching a target an obstacle avoidance behavior has to be active for safety reasons while the performance is improved if the robot tries to approach the goal at the same time as it is avoiding obstacles. This calls for a fused, coordinated control scheme [2],[3].

2.1 Obstacle Negotiation

The specific problem that will be investigated in this article is how to move a robot between two points. This *point-to-point motion* should be done so that

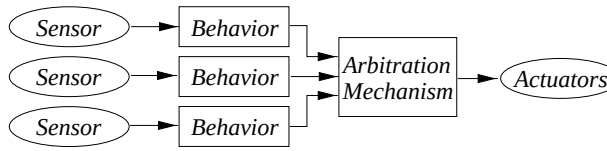


Fig. 1. Block diagram of the behavior based control architecture.

the detection of an obstacle results in a repulsive potential field, acting on the platform when the robot is closer to the obstacle than a desired safety distance, d_{OA}, where the subscript stands for obstacle avoidance. This behavior is an example of a so called *reactive* obstacle avoidance behavior. The word reactive, a commonly used one in the robotics community, is used here since the behavior can be thought of as a reflex. When the robot moves too close to an obstacle, it is forced to change the motion in order to avoid hitting the obstacle. This is a reasonable safety strategy since the robot may be moving around in a highly unstructured world, where the occurrence of unpredicted, or unmodeled obstacles is very likely.

We now assume that we have direct access to the robot's longitudinal velocity, v, at the same time as the heading of the robot, ϕ, can be controlled directly as

$$\dot{\phi} = \omega.$$

Furthermore, if the sonars on the robot, with center of gravity at (x, y) and heading ϕ, detect a point-obstacle at (x_{ob}, y_{ob}) that is closer to the robot than d_{OA}, the reactive control response will be given by a vector field acting on the robot as

$$\omega = C_{OA} W_{OA}(d)(\tilde{\phi} - \phi), \tag{1}$$

$$d = \sqrt{(x - x_{ob})^2 + (y - y_{ob})^2},$$

where and

$$W_{OA}(d) = \begin{cases} \frac{1}{d^2} \text{ if } d < d_{OA} \\ 0 \text{ if } d \geq d_{OA}, \end{cases}$$

$$\tilde{\phi} = \pi + \operatorname{atan2}(y_{ob} - y, x_{ob} - x), \tag{2}$$

as seen in Figure 2. Here, C_{OA} is just a constant weight, and d_{OA} is the fixed distance from the obstacle where the behavior becomes active.

Since a real, extended obstacle cannot be considered to be a point, in the actual implementation of the avoidance behavior, the desired heading needs to be calculated as the orientation of the sum of the weighted vectors that each individual sonar reading contributes with. For a Nomad 200, that is going to be our experimental platform, this corresponds to taking the sum over 16 elements since the Nomad is equipped with 16 ultrasonic sensors.

A standard kinematic model of the mobile robot [1] gives that

$$\begin{aligned} \dot{x} &= v \cos \phi \\ \dot{y} &= v \sin \phi, \end{aligned} \tag{3}$$

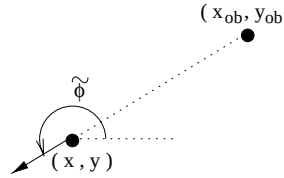


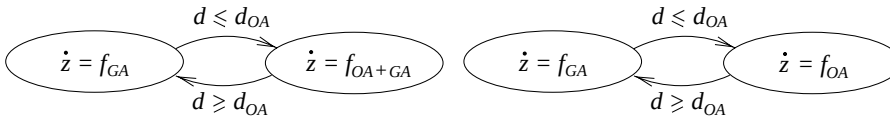
Fig. 2. The general idea behind a repulsive obstacle avoidance behavior.

and if we set $z = (x, y, \phi)$, we let $\dot{z} = f_{OA}(z)$ denote the full state, closed-loop obstacle avoidance behavior where we initially let v be constant.

2.2 Hybrid Automata

When adding a goal attraction behavior, defined in the same way as the obstacle avoidance behavior except that we now have an attractive instead of a repulsive field, we get two different possible hybrid automata for describing the situation. This depends on whether the two behaviors are active simultaneously or not, as seen in Figure 3. If one chooses to work with fused, concurrently active behaviors, then different controllers affect the system simultaneously, resulting in a smooth overall performance [11]. But in that case, however, the system does not correspond to an automaton where each node represents a single behavior. This would make the automata approach meaningless since we would then just "hide" all of the difficulties that the complex control system gives rise to in the individual nodes of the automaton.

On the other hand, the other possible solution to the coordination problem, corresponding to hard switches between the different behaviors, has the major disadvantage that it both affects the performance in a negative way, not allowing for the smooth performance that fused behaviors produce, and that it increases the risk of introducing chattering into the system. Therefore our idea is to impose hard switches on the behavior based system in such a way that we can model



(a) Fused behaviors (b) Hard switches

Fig. 3. The two possible goal attraction and obstacle avoidance automata. Here, d_{OA} is the fixed distance from the obstacle where the obstacle avoidance behavior becomes active.

each behavior as a node in an automaton, at the same time as we want to avoid the negative, chattering effects that such an approach could potentially give rise to. This will be done by adding nodes to the automaton as a way of regularizing it, and in what follows we will show that even though we introduce hard switches, the performance is not affected much when using a regularized automaton instead of fused behaviors. In other words, what we want to do is to remove some of the so called *Zeno*[1] properties of the system. What this corresponds to is a hybrid system that exhibits an infinite number of discrete transitions in finite time.

Even though the main focus in this article is not going to be on hybrid automata theory, we need to include some initial definitions. This is necessary in order to be able to state what we mean by a Zeno hybrid automaton as well as to capture the hybrid aspects of a behavior based robotic system.

The following brief definitions are based on [6],[8],[14].

Definition 1 (Hybrid Automaton). *A hybrid automaton is considered to be a collection* (Q, X, I, f, E) *where* Q *and* X *are sets of discrete and continuous variables respectively.* I *is a set of initial states, while* f *describes the continuous and* E *the discrete evolution of the states.*

A discrete state combined with the continuous dynamics connected to that state will be referred to as a *node* in the automaton. The general idea behind this construction can be seen in Figure 4.

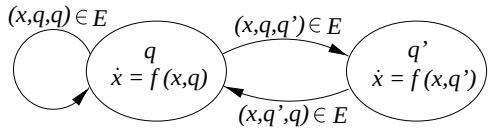


Fig. 4. The basic structure of a hybrid automaton.

Definition 2 (Hybrid Time Trajectory). *A hybrid time trajectory* τ *is a finite or infinite sequence of intervals of the real line,* $\tau = \{I_i\}$, $i \in \mathbb{N}$, *satisfying the following conditions:*

- I_i *is closed, unless* τ *is a finite sequence and* I_i *is the last interval in which case it can be right open.*
- *Let* $I_i = [\tau_i, \tau_i']$. *Then for all* i, $\tau_i \leq \tau_i'$ *and for* $i > 0$, $\tau_i = \tau_{i-1}'$.

This should be interpreted as the times at which we arrive (τ_i) and leave (τ_i') a specific node in the automaton.

[1] The name Zeno refers to the philosopher Zeno of Elea (500–400 B.C.), whose major work consisted of a number of famous paradoxes. They were designed to explain his view that the ideas of motion and evolving time lead to contradictions. An example is Zeno's Second Paradox of Motion, in which Achilles is racing against a tortoise.

Note that hybrid time trajectories can extend to infinity if τ is an infinite sequence or if it is a finite sequence ending with an interval of the form $[\tau_N, \infty)$.

Definition 3 (Execution). *An execution χ of a hybrid automaton H is a collection $\chi = (\tau, q, x)$, satisfying*

- *Initial Condition:* $(q(\tau_0), x(\tau_0)) \in I$.
- *Discrete Evolution:* $(x(\tau'_{i-1}), q(\tau'_{i-1}), q(\tau_i)) \in E$, *for all* i.
- *Continuous Evolution: for all i with $\tau_i < \tau'_i$, x and q are continuous over $[\tau_i, \tau'_i]$ and for all $t \in [\tau_i, \tau'_i)$, we have* $\frac{d}{dt}x(t) = f\big(q(t), x(t)\big)$.

Furthermore, an execution $\chi = (\tau, q, x)$ is called infinite, if τ is an infinite sequence, or $\sum_i(\tau'_i - \tau_i) = \infty$. We use $\mathcal{H}_{(q_0,x_0)}$ to denote the set of all infinite executions of H with initial condition $(q_0, x_0) \in I$. An execution is *admissible* if $\sum_i(\tau'_i - \tau_i) = \infty$, and it is *Zeno* if it is infinite but not admissible. For a Zeno execution $\chi = (\tau, q, x)$ we define the Zeno time as $\tau_\infty = \sum_i(\tau'_i - \tau_i) < \infty$. What this means is that the hybrid system makes an infinite number of discrete transitions in finite time, $[\tau_0, \tau_\infty]$, and we finally state the following definition.

Definition 4 (Zeno Hybrid Automaton). *A hybrid automaton H is called Zeno, if there exists $(q_0, x_0) \in I$ such that $\mathcal{H}_{(q_0,x_0)}$ contains a Zeno execution.*

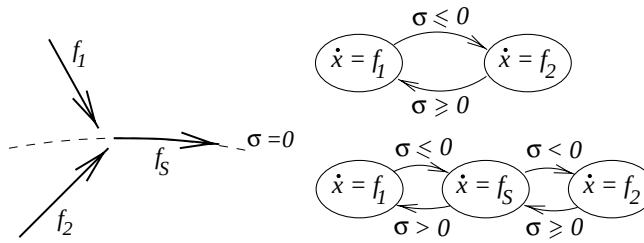
2.3 Regularization

It is clear that a Zeno hybrid automaton has the undesirable property that it blocks time. For the type of automata that we will encounter here, the infinite number of discrete transitions, made in finite time, is caused by the fact that the underlying system that the automaton tries to model is a switched system that exhibits sliding in the sense of Filippov [7]. They thus form a special class of Zeno hybrid automata since they, in theory, make an infinite number of transitions in zero time.[2] The underlying, switched systems have continuous flows that point toward the switching surface, resulting in a new, induced flow on that surface. In these cases, the automaton can be regularized by the introduction of a new node with the continuous flow given by the Filippov solution [8],[15]. The general idea behind this construction can be seen in Figure 5.

If we now assume that C_{OA} in (1) is large enough so that the heading of the robot can be considered to be more or less instantaneously driven to its desired configuration, the hybrid automaton in Figure 3(b) can admit Zeno executions. This obvious fact is best illustrated by Figure 6, where the extra node that needs to be added in order to regularize the automaton can easily be identified as well. The extra node is just a node containing the sliding dynamics that is defined on the boundary between the two behaviors.

When an obstacle is closer to the robot than d_{OA}, the obstacle avoidance behavior becomes active. Since the repulsive potential field from that behavior

[2] The other class of Zeno automata has a slightly more complex dynamics. Here the automaton changes nodes faster and faster, with the jump times converging to the Zeno time, τ_∞ [8].



(a) The sliding solution.

(b) The original and the regularized automaton.

Fig. 5. Regularization of a Filippov type Zeno hybrid automaton.

is orthogonal to the surface on which the behavior becomes active, the sliding solution is just

$$f_S = \alpha f_{OA} + (1 - \alpha) f_{GA},$$

where GA stands for goal attraction, and $\alpha \in [0, 1]$ is chosen so that $f_S \perp f_{OA}$. Adding this type of information about the different behaviors makes it possible to generate the extra node in the automaton automatically. It furthermore suggests that our method would scale when more that two behaviors affect the motion of the robot, as long as an automatic procedure for designing the sliding solutions could be identified for the new behaviors as well.[3]

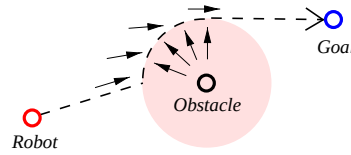
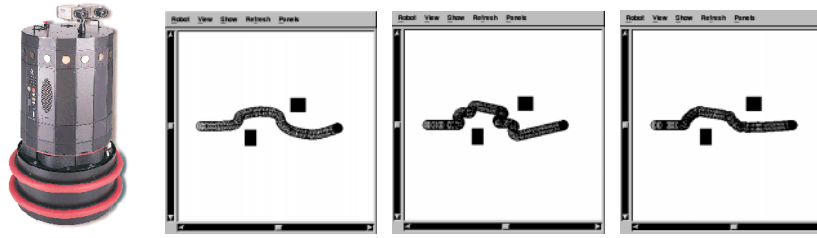


Fig. 6. Goal attraction together with obstacle avoidance results in a Filippov type Zeno automaton. The grey region around the obstacle corresponds to the region where obstacle avoidance is active. The arrows correspond to the different vector fields that are acting on the robot.

The assumption about instantaneous heading control is obviously a simplification but it still gives a model that is rich enough to capture the, from our point of view, relevant phenomena. In fact, in real life we have a possibility of chattering that here reveals itself as a Zeno execution.

[3] This typically depends on whether we have access to a geometric description of the switching surface or not.



(a) Nomad 200 (b) Fused behaviors (c) Hard switches (d) Regularized hybrid automaton

Fig. 7. Simulation of b) fused behaviors, c) hard switches, and d) a regularized automaton on the Nomad simulator, the `Nserver`.

The regularized point-to-point automaton was implemented and tested on a Nomad 200 mobile robot. In Figure 7, the results from running the system on the `Nserver`, the Nomad simulation package, can be seen. In (7b) fused behaviors are displayed, resulting in a smooth movement around the obstacle, while the chattering solution in (7c) corresponds to hard switches. The reason why we do not have sliding in this case is due to the part of the dynamics of the robot that was ignored in the analysis. It is still clear that from a performance perspective, (7c) is an unsuccessful design. In (7d) the result from using a regularized automaton can be seen, and even though we only have one behavior active at a time, the performance is satisfactory.

3 Controller Design

Given the reactive obstacle avoidance behavior from the previous section, the main question that we want to address here is: *How do we construct an appropriate approach target behavior?* Obviously, we can do better than to just use an attractive potential field, and it will turn out that our automata approach allows us to explicitly deal with safety and optimality.

What we want to do is to produce a robot behavior that satisfies the safety specifications at the same time as the solution is close to optimal with respect to a given performance evaluation functional, and a first formulation, inspired by [19], of what we want to accomplish could be the following. If we let our admissible controls be $u \in \mathcal{U}$, and define a safety functional

$$\mathcal{J}_s(u) = \min_{t \geq 0} \left\{ (x(t) - x_{ob})^2 + (y(t) - y_{ob})^2 \right\}, \tag{4}$$

where the dependence on the control, u, is given implicitly by the controlled system dynamics from the previous section. The set of controls, $\mathcal{U}_s(C)$, that

make the robot move at least a distance C away from the obstacle, can thus be defined as

$$\mathcal{U}_s(C) = \{u \in \mathcal{U} \ : \ \mathcal{J}_s(u) \geq C\}\,. \tag{5}$$

It should be mentioned that both $\mathcal{J}_s$ and $\mathcal{U}_s$ depend on the robot's initial position, but for the sake of notational simplicity we leave that out from the definitions.

The next step is to define another cost functional that penalizes high curvature of the chosen path. This is a reasonable performance criterion since it penalizes paths that make the robot move in sudden, abrupt ways. Furthermore, this smoothness objective gives a trajectory that a robot has good chances of following when it is governed by physical limits on what signals the actuators can actually track. In some other situations, such as when a mobile manipulator is asked to carry a cup of coffee, the smoothness of the curve is absolutely crucial and is obviously of key importance to a successful, "non-spilling" execution of the task.

The idea now is to choose the control candidates for minimizing this new performance functional from the set of safe controls, $\mathcal{U}_s(C_s)$, where C_s is our preferred safety margin.

Unfortunately it turns out that this is a very hard problem to solve numerically (not to mention analytically) [19], which implies that in this formulation, it is not suitable for situations where on-line computations are necessary. However, the underlying approach could suggest a way for producing a solution to the obstacle negotiation problem that is both safe, computationally feasible, and makes the system behave in a satisfactory way with respect to keeping the curvature of the produced path small.

The main idea is that instead of focusing on the hard optimal control problem, we should concentrate on just producing optimal (or close to optimal) geometric trajectories that lead around the obstacle. This way we do not have to deal with the actual kinematics of the robot in the optimization formulation. Instead we add the kinematics when we track the produced path. This means that we cannot be sure that we actually find the optimal controller, but rather that we find one that is reasonably close to the optimal one as long as we have a good enough trajectory tracker.

The desired overall behavior that these heuristics give rise to (under the assumption of perfect tracking), together with the corresponding automaton, is depicted in Figure 8.

3.1 Path Planning

One first observation is that for a path produced by a scalar function $y_d = f(x_d)$, the curvature is given by

$$\kappa(x_d) = \frac{f''(x_d)}{(1 + f'(x_d)^2)^{3/2}}, \tag{6}$$

where the subscript d stands for the desired robot position.

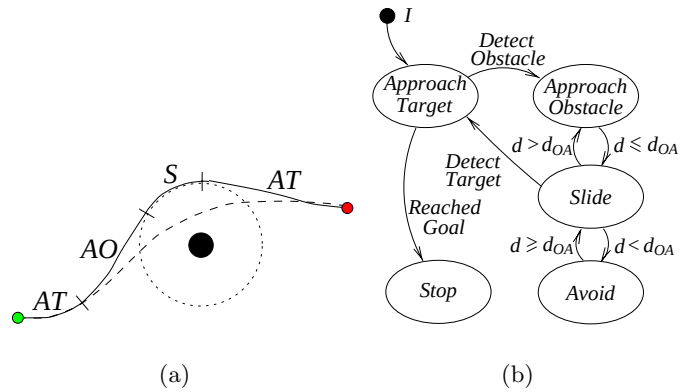


Fig. 8. In the left figure, an optimal path is planned and followed by an Approach Target behavior until an obstacle is detected. Then an Approach Obstacle behavior follows another path to the region where the regularized sliding behavior becomes active. When the target can be reached by an optimal path, not intersecting the safety region around the obstacle (called Detect Target in the right figure), Approach Target becomes active again. In the right figure, the corresponding automaton is depicted.

Thus, if we minimize $f''(x_d)^2$ instead of $\kappa(x_d)^2$ we make $\kappa(x_d)^2$ small automatically, which is a desired feature, as seen in the previous paragraph.

Since we, by following this proposed route, minimize the L^2-norm of the second derivative, the resulting curve will be a *cubic spline*. This is a fortunate fact since it means that we will not be forced to relay on extensive world information or to do any heavy computations on-line which tends to be the case when more sophisticated planning algorithms are used [12],[13],[18]. It is thus an almost trivial task to generate the splines that connect the robot and the target in the approach target behavior, and the robot and the obstacle in the approach obstacle behavior, as seen in Figure 8.

3.2 Tracking

We now have an on-line method for producing low curvature paths around detected obstacles, and hence our next task is to find a good tracking algorithm so that the robot follows the proposed path robustly.

We let the general reference path, parameterized by s, be given by

$$\begin{matrix} x_d = p(s) \\ y_d = q(s) \end{matrix} \quad 0 \leq s \leq s_f, \tag{7}$$

where the idea is to let the motion of the reference point be governed by a differential equation containing error feedback. It can be viewed as a combination of

the conventional trajectory tracking, where the reference trajectory is parameterized in time, and a dynamic path following approach [17], where the criterion is to stay close to the geometric path, but not necessarily close to an *a priori* specified point at a given time. This approach makes our algorithm robust to measurement errors and external disturbances since, if both the tracking errors and disturbances are within certain bounds, the reference point moves along the reference trajectory while the robot follows it within a prespecified look-ahead distance. Otherwise, the reference point should slow down and "wait" for the robot.

Our control objectives are

$$\begin{array}{l} \limsup_{t\to\infty} \rho(t) \leq \epsilon_\rho \\ \limsup_{t\to\infty} |\phi(t) - \phi_d(t)| \leq \epsilon_\phi, \end{array} \tag{8}$$

where ϵ_ρ and ϵ_ϕ are positive numbers that can be made arbitrarily small, $\rho(t) = \sqrt{(x_d - x)^2 + (y_d - y)^2}$, where (x,y) is the actual position of the robot, and ϕ and ϕ_d are actual and desired robot orientations.

From (7) we directly get that $\dot{x}_d = p'(s)\dot{s}, \dot{y}_d = q'(s)\dot{s}$, which implies that if the robot would track the path perfectly, we would have

$$\dot{s} = \frac{p'(s)}{p'^2(s) + q'^2(s)}\dot{x} + \frac{q'(s)}{p'^2(s) + q'^2(s)}\dot{y}, \tag{9}$$

since this corresponds to $\dot{x} = \dot{x}_d$ and $\dot{y} = \dot{y}_d$. On the other hand, (9) does not contain any position error feedback, which is important for the robustness. Therefore we propose our dynamics for the reference point as follows:

$$\dot{s} = \frac{ce^{-\alpha\rho}v_0}{\sqrt{p'^2(s) + q'^2(s)}}, \tag{10}$$

where v_0 is the desired speed at which one wants the vehicle to track the path, and α and c are appropriate, positive numbers.

We now let our control algorithm be as follows:

$$\begin{array}{l} v = \gamma\rho\cos(e_\phi) \\ \omega = ke_\phi + \dot{\phi}_d, \quad k > 0, \end{array} \tag{11}$$

where both γ and k are positive, $e_\phi = \phi_d - \phi$, and $\phi_d = \arctan2(y_d - y, x_d - x)$.

In [4],[5] it was shown that for the platform model (3), governed by the control (11), the steady state tracking error, ρ, can be made as small as one wants while ϕ tends to ϕ_d exponentially. Furthermore, in steady state we have that $v \approx v_0$. Thus we, by using the control law (11), meet the control objectives defined in (8).

We thus have a way of both producing and tracking paths, and we now combine these two together into the path following behavior that moves the robot safely around the obstacles at the same time as its executed trajectories are not too far from optimal with respect to curvature. As seen in Figure 9, where real experimental data are displayed, the method seems to work well.

(a) (b)

Fig. 9. The results from implementing our ideas on the Nomad 200 can be seen. The reason why the sonar readings seem rather inaccurate is due to the fact that the robot has some drift in the odometry at the same time as the sonar resolution is rather coarse.

4 Implementation

There is a need to be able to define the hybrid automata in a structured and systematic way, making it easy to reuse and reorder nodes in different configurations. Therefore, at the Centre for Autonomous Systems (CAS) [2] at KTH, a programming environment for doing mobile manipulation[4] within the hybrid automata framework has been developed [16]. It is called the *MMCA*, the Mobile Manipulation Control Architecture, and the core of the MMCA is an engine that executes hybrid automata, where, as mentioned, the nodes corresponds to different behaviors. The architecture is designed to be open and allows the user to experiment with the contents of the behaviors freely, e.g. internal representations and algorithms, as long as the behaviors contain:

(i) A function returning the desired state (in our case joint angles and platform pose)

(ii) Conditions for when to make the discrete transitions

A program written in the MMCA language begins with a specification of the initial node. Then all the nodes in the automaton are listed, where each node is specified by name, type, parameters and transitions. The transitions refer to the other nodes, or to itself, and the type of a node determines its functionality, such as what type of controller it is using, and it also defines which parameters or initial values that can be passed to the node.

A sample file that defines the task of opening a door might look like

```
INTERFACE = Puma560_XR4000;
INITNODE = Approach;
BEGIN
   NAME = Approach;
```

[4] From our point of view, this simply means that the mobile, behavior based platform has been augmented by the addition of a robotic arm, mounted on top [9].

```
   TYPE = Visual_Servo;

   Object = Door_Handle;    % Servo on a door handle
   TRANSITION[End_Position] = Grasp;
END
BEGIN
   NAME = Grasp;
   TYPE = Grasp_Object;

   Object = Door_Handle;    % Grasp a door handle
   TRANSITION[Got_Grip]  = Pull;
   TRANSITION[Lost_Grip] = Approach;
END
BEGIN
   NAME = Pull;
   TYPE = Follow_Arc;

   Radius = 0.8;            % Estimate of the arc radius
   Angle  = 90;             % Open door 90 deg.
   TRANSITION[Ready] = End; % Terminates the control cycle
END
```

5 Conclusions

In this article, it is shown that a behavior based control system can be modeled as a hybrid automaton, where each node corresponds to a distinct robot behavior. In order to achieve this, we have to impose hard switches on the transitions between the different behaviors, resulting in a potentially chattering overall behavior. We furthermore show how regularization techniques can be used to solve this problem by adding extra nodes to the automaton. Those extra nodes correspond to the sliding dynamics on the boundary between the different behaviors. The performance aspect of this approach is verified experimentally on a Nomad 200 mobile platform.

We also propose a heuristic method for designing reach target behaviors in such a way that questions concerning safety and optimality can be addressed explicitly. Our proposed method is based on a combination of path planning and trajectory tracking techniques, placing it in the deliberative part of the behavior based control architecture spectrum. Furthermore, we show that this approach works well in practice on our experimental platform.

We conclude the article with a brief presentation of a programming environment, the MMCA, for defining hybrid automata in a systematic and structured way.

Acknowledgments

The author would like to thank Karl Henrik Johansson, John Lygeros, and Shankar Sastry for valuable comments about the regularization aspects of hybrid automata. He would also like to thank Xiaoming Hu and Henrik Christensen for their ideas on autonomous robotics, and Lars Petersson for helping developing the MMCA programming environment.

References

1. J. Ackermann. *Robust Control*. Springer-Verlag, London, 1993.
2. M. Andersson, A. Orebäck, M. Lindström, and H.I. Christensen. *Intelligent Sensor Based Robotics*. Ch. ISR: An Intelligent Service Robot, Lecture Notes in Artificial Intelligence, Heidelberg: Springer Verlag, 1999.
3. R.C. Arkin. *Behavior-Based Robotics*. The MIT Press, Cambridge, Massachusetts, 1998.
4. M. Egerstedt, X. Hu, and A. Stotsky. Control of a Car-Like Robot Using a Virtual Vehicle Approach. Proceedings of the *37th IEEE Conference on Decision and Control*, pp. 1502–1507, Tampa, Florida, USA, Dec. 1998.
5. M. Egerstedt, X. Hu, and A. Stotsky. A Hybrid Control Approach to Action Coordination for Mobile Robots. Proceedings of *IFAC'99:14th World Congress*, Beijing, China, Jul., 1999.
6. M. Egerstedt, K. Johansson, J. Lygeros, and S. Sastry. Behavior Based Robotics Using Regularized Hybrid Automata. Proceedings of *CDC'99*, Phoenix, Arizona, Dec, 1999.
7. A.F. Filippov. *Differential Equations with Discontinuous Righthand Sides*. Kluwer Academic Publishers, 1988.
8. K. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry. Regularization of Zeno Hybrid Automata. *Systems and Control Letters*, 1999. Accepted for publication in 1999 Special Issue on Hybrid Systems.
9. O. Khatib, K.Yokoi, K.Chang, D.Ruspini, R.Holmberg, A.Casal and A.Baader: Force Strategies for Cooperative Tasks in Multiple Mobile Manipulation Systems. *International Symposium of Robotics Research*, Munich, October 1995.
10. D. Kortenkamp, R.P. Bonasso, and R. Murphy, Eds. *Artificial Intelligence and Mobile Robots*. The MIT Press, Cambridge, Massachusetts, 1998.
11. J. Košecká: *A Framework for Modeling and Verifying Visually Guided Agents: Design, Analysis and Experiments*. Dissertation, Grasp Lab, March 1996.
12. B. Krogh and C. Thorpe. Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles, Proceedings of the *1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, pp. 1664-1669, 1986.
13. J.C. Latombe. *Robot Motion Planning*, Kluwer Academic Publishers, 1991.
14. J. Lygeros, C. Tomlin, and S. Sastry. Controllers for Reachability Specifications for Hybrid Systems. *Automatica*, Vol. 35, No. 3, March 1999.
15. J. Malmborg. *Analysis and Design of Hybrid Control Systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, May 1998.
16. L. Petersson, M. Egerstedt, and H.I. Christensen. A Hybrid Control Architecture for Mobile Manipulation. Proceedings of the *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Kyongju, Korea, Oct. 1999.
17. N. Sarkar, X. Yun, and V. Kumar. Dynamic Path Following: A New Control Algorithm for Mobile Robots. *Proceedings of the 32nd Conference on Decision and Control*, San Antonio, Texas, Dec. 1993.
18. A. Stenz. Optimal and Efficient Path Planning for Partially Known Environments, Proceedings of the *1994 IEEE International Conference on Robotics and Automation*, 1994.
19. C. Tomlin, G. Papas, J. Košecká, J. Lygeros, and S.S. Sastry. Advanced Air Traffic Automation: A Case Study in Distributed Decentralized Control, *Control Problems in Robotics*, Lecture Notes in Control and Information Sciences 230, Springer-Verlag, London, 1998.