Behavior-Based Control of a Non-Holonomic Robot in Pushing Tasks

Rosemary Emery and Tucker Balch

Abstract

We describe a behavior-based control system that enables a non-holonomic robot to push an object from an arbitrary starting position to a goal location through an obstacle field. The approach avoids the need for maintaining an internal model of the target and obstacles and the potentially high computational overhead associated with traditional path planning approaches for non-holonomic robots. The motor schema-based control system was prototyped in simulation, then evaluated empirically on mobile robots. The approach was demonstrated in two pushing tasks: box pushing and ball dribbling. Using the same generalized control system, a robot is able to successfully push a box through a static obstacle field and dribble a ball into a goal during RoboCup soccer matches. In both cases the robot is able to react quickly and recover from situations where it loses control of its target either due to its own motion or interference with others. Quantitative experimental results for reliability in box pushing are included.

I. Background and Related Work

Pushing is the basis for many useful tasks. In a construction scenario, for instance, bulldozer robots are essentially performing a pushing task in which the target object is soil [14]. Pushing is also important in other materials handling tasks such as assembly, construction and cleaning. For pushing, a robot does not require lifting or gripping mechanisms and therefore is more tolerant to load size and shape. Also, heavy, unliftable, objects can be transported by multiple robots coordinating their pushing efforts [12], [13].

Several of the "sub-skills" necessary for pushing are also useful in other tasks. For example, pushing requires a robot to align itself properly at the target with respect to a goal location and then maintain control of that object while travelling to the goal. The alignment phase of pushing can be used to properly position a robot for picking up a target. If the robot is non-holonomic, this alignment phase is non-trivial.

R. Emery and T. Balch are with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213. E-mail: {remery, trb}@cs.cmu.edu .

A number of researchers have investigated path and push planning for non-holonomic robots; however, these solutions tend to address static environments and they are computationally more expensive than our approach [1], [10], [11]. Additionally, traditional path planning approaches for a pushing task may require accurate models of the environment or other global information that is not easily available to a robot in real applications.

One alternative to path-planning is a behavior-based approach. This family of methods closely tie perception to action which makes them appropriate for dynamic environments or environments for which no *a priori* world model exists. In this work we use a motor schema-based approach which combines behavioral building blocks using vector summation similar to potential field navigation methods [2], [10].

Ours is not the first behavior-based approach to box pushing. Parker, and separately, Mataric, for instance, have investigated coordinated multi-robot box pushing [13], [12]. However, ours is the first to address the problem of pushing while avoiding obstacles and simultaneously accounting for the non-holonomic constraints of a tricycle-like robot.

II. The Problem

A. The Robot

The robots used in this work were constructed at the CMU MultiRobot Lab as part of a project to build inexpensive, autonomous robots for the study of multirobot systems operating in dynamic and uncertain environments. The underlying mechanical platform is a commercially available non-holonomic robot composed of two parts: a differentially steered motorized drive unit and a trailer. The trailer serves as a mount for an on-board computer, image capture system and power supply (left side of Figure 1). The drive unit is roughly 42cmx23cm and the trailer is 43cm long and is mounted to the center of the drive unit.

While the differentially steered drive unit can easily perform lateral movements by rotating, driving and then rotating back to its original heading, when paired with the trailer the combined vehicle cannot. To avoid the robot's field of view becoming blocked

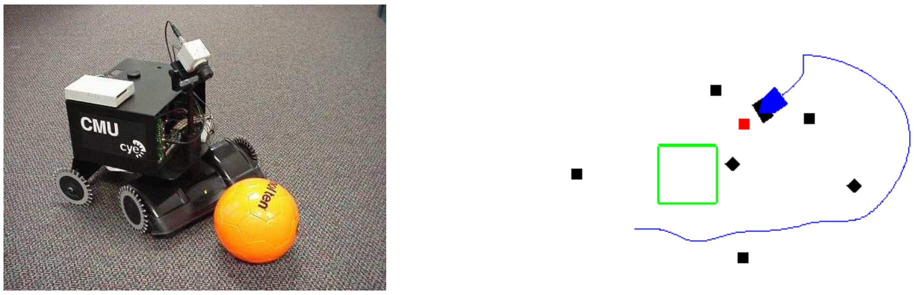

Fig. 1. Two pushing tasks. Left: The Minnow robot dribbling a soccer ball. Right: Sample box pushing task set up. The robot (trajectory represented by black line) has lined up behind the block facing the goal and is ready to start pushing the block to the goal location indicated by the open square.

by the trailer and to reduce impairment in pushing, the robot's low level software prevents the drive unit from turning in place such that it is facing the trailer (the trailer constraint). Thus, in order to achieve an arbitrary heading the robot must, at some point, drive forward with a minimum turning radius. A useful controller for this robot must somehow account for these constraints.

The robot is fully autonomous with a Pentium MMX processor-based single board computer that utilizes wireless Ethernet for communicating with other robots and data logging. The robot uses color segmentation of the images it receives to identify objects in its world [8].

B. The Task

We define the pushing task for a robot as locating an object (usually a box), and pushing it to a goal location through an obstacle field. The robot and target's starting positions are arbitrary and the obstacles may be placed such that it is necessary for the robot to navigate through them, both to get to the target and while pushing the target to the goal. Because the robot has a limited field of view, and it is not provided the location of the target object *a priori*, it must search the environment to find it. Additionally, the task is complicated by the non-holonomic constraints of our robot. An example starting configuration is presented on the right in Figure 1.

The obstacles in the pushing task are static or moving based upon the task's context (e.g. box pushing or soccer). Additionally, the goal location may be specified as a a location in Cartesian coordinates or as a visual target. In the first case the robot uses odometry to determine its position relative to the goal location and in the second, its vision system is used to locate the goal as a position relative to its current position.

Unlike a holonomic robot, a non-holonomic robot cannot achieve all possible configuration states directly from its current state. This complicates the pushing task because the robot must find a path, preferably near optimal, from its current state to the state that aligns it with the target object and goal location. Furthermore, the presence of obstacles can reduce the robot's room for maneuverability and thus prevent a non-holonomic robot from finding a path that allows it to achieve alignment. Fully holonomic robots, however, are not so constrained and are therefore more likely to get to the target object aligned correctly.

III. Approach

As discussed above, the pushing task can be decomposed into several subtasks. The robot must search for a target, acquire it and then deliver it to the goal location. The robot's program for accomplishing the task can be represented as a finite state machine diagram (Figure 2). In this approach, referred to as *perceptual sequencing*, each state corresponds to a suite of activated behaviors for accomplishing that step of the task [4]. Transitions between states, called *perceptual triggers*, are initiated by real-time perception.

In the pushing task, the first state is for searching for the target object. Once the robot locates the target it starts to move towards it, using a reactive approach to maneuver such that it will line up behind the target facing the goal location (this step is critical for the non-holonomic robot). When the robot is lined up reasonably well behind the target it will switch to a pushing behavior where it moves towards the goal while keeping the target under its control. Even so, the robot may lose control of the target and even possibly lose sight of the target. In the latter case the robot will return to the searching state; panning until it sees the target again. If the robot can still see the target but has lost control of it, it returns to the acquiring state and will re-maneuver itself until it is lined up behind the target. These state transitions will carry on until the robot has successfully delivered the target to the goal location. At this point the robot will back up and watch the target, going back into the scanning behavior should the target be removed from view.

The finite state machine controlling the robot was implemented using the Clay library of the TeamBots architecture [5], [6]. TeamBots is a Java-based collection of application programs and packages for multi-agent robotics research. The Clay library is a group of Java classes which are easily combined to create motor schema-based control systems [2]. At the basis of Clay are perceptual and motor nodes (schemas) that are combined to create behavioral sequences. Perceptual nodes take information from the robot's sensors such as the location of objects of interest and obstacles. These nodes are embedded in motor schemas to produce vectors representing the desired trajectory of the robot.

Behavioral assemblages are formed by combining one or more motor schemas. In this work the motor schemas were combined using linear superposition; each schema is assigned a weight indicating its contribution to the overall behavior. The approach is similar to potential field navigation strategies; however, the complete field is never calculated – only the vector at the robot's position. Behavioral assemblages form the basis of the finite state machine with each behavior assemblage corresponding to a state and perceptual nodes triggering transitions between those states. Therefore the three states of Search, Acquire and Deliver shown in Figure 2 are equivalent to three distinct behavioral assemblages.

This control level computes desired robot headings and the velocities at which the robot should achieve them. In a lower software level these headings are transformed into appropriate motor velocities for each of the drive unit's two wheels. The low level software only permits the robot to turn in place a certain amount, after which it must complete its turn by driving both wheels forward. The speed of each wheel depends on how close the robot's current heading is to the desired heading (this is to accommodate the constraints imposed by the trailer). As there is no encoder on the trailer, the low level software is also responsible for maintaining an estimation of the trailer's angle with respect to the robot and this is done using kinematic equations of motion for a tricycle [9].

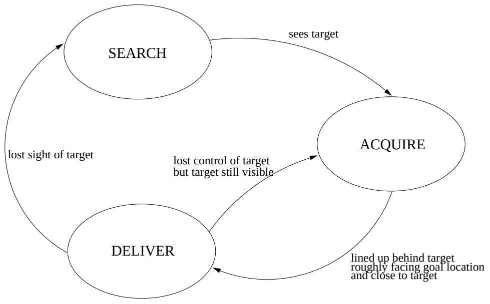

Fig. 2. Finite state machine used for behavior-based pushing tasks.

It is important to note that the vector computed by a behavioral assemblage is interpreted as a heading along which the robot should be travelling. The velocity at which the robot should achieve that heading is calculated separately. For the Search behavioral assemblage the robot's speed is set at zero, while for the Acquire and Deliver assemblages the robot's speed is fixed unless the robot is very close to the target or goal, in which case it slows proportionally as its distance to the object in question decreases.

A. Motor Schemas for the Pushing Task

We will now introduce the primitive components of the robot's behaviors (the motor schemas) and their mathematical formulations. Next we describe how the motor schemas are combined to form behavioral assemblages for accomplishing each step in the task. Finally the overall, sequenced behavior is presented.

Each motor schema computes a *direction* and *magnitude* corresponding to which way the robot should move, and how critical it is that it move in that direction. Because the schemas are combined using vector superposition, the magnitude directly impacts the influence a particular schema has on the final direction of the robot.

A.1 Scan

The Scan motor schema causes the robot to rotate back and forth in place. In general, at every time step the Scan motor schema creates a vector equivalent to a rotation relative to the robot's current heading. The robot will rotate in place in an attempt to achieve the current desired heading; however, if lower level software determines that the robot cannot continue to turn in that direction without violating the trailer constraint, it stops the rotation and a perceptual schema is activated to inform the Scan motor schema that the current heading vector is invalid. The Scan motor schema then reverses its turn direction. While active, this schema results in repeated clockwise rotation followed by counter-clockwise rotation.

The Scan motor schema is parameterized by X, the number of degrees to rotate in each direction. The mathematical formulation for this schema is as follows:

$$
\begin{aligned}
V_{\text{scan}} &= \text{current heading +/- } X\text{-degrees} \\
\|V_{\text{scan}}\| &= 1.0
\end{aligned}
$$

A.2 Go-To-Target

The Go-To-Target motor schema is implemented as a linear attraction. The magnitude of the attraction varies with distance to the target; 1.0 outside of a controlled zone, decreasing linearly from 1.0 to 0.0 at the dead zone's boundary, and finally 0.0 within the dead zone. The parameters C_1 and D_1 (shown in Figure 3) are used to specify the radii of the controlled and dead zones respectively. Mathematically:

$$
\begin{aligned}
V_{\text{go-to-target}} &= \text{vector from center of robot} \\
&\quad\ \text{to target object} \\
\|V_{\text{go-to-target}}\| &= \begin{cases} 1 & \text{for } r > C_1 \\ \frac{r-D_1}{C_1-D_1} & \text{for } D_1 < r \leq C_1 \\ 0 & \text{for } r \leq D_1 \end{cases}
\end{aligned}
$$

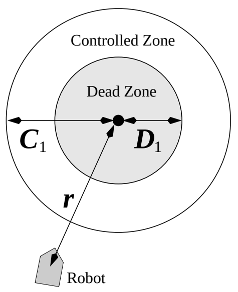

Fig. 3. Parameters used in the calculation of Go-To-Target.

A.3 Swirl-Obstacles

At all times the robot is avoiding obstacles. Avoidance is implemented using circumnavigation or "swirling" around each obstacle (Figure 4). The appropriate direction for circumnavigation depends on whether the robot is attempting to reach the target (acquire) or the goal location (deliver) [6]. The Swirl-Obstacles motor schema creates a vector perpendicular to the line from the robot to each obstacle it detects in the appropriate direction. The magnitude of each of these vectors is zero beyond a controlled zone and infinite within a dead zone. Between the two it increases linearly until it reaches a maximum value at the dead zone boundary. The vectors corresponding to each obstacle are then summed to form the output of this schema.

The parameters for the Swirl-Obstacles schema are C_2 and D_2, the controlled and dead zone radii (defined similarly to C_1 and D_1 as shown in Figure 3). The mathematical formulation for Swirl-Obstacles is:

$$
\begin{aligned}
V_{\text{i}} &= \text{vector from center of target to} \\
&\quad\ \text{obstacle rotated +/- 90 degrees} \\
&\quad\ \text{such that rotation sweeps through} \\
&\quad\ \text{object swirling with respect to} \\
\|V_{\text{i}}\| &= \begin{cases} 0 & \text{for } r > C_2 \\ \frac{C_2-r}{C_2} & \text{for } D_2 \leq r \leq C_2 \\ \infty & \text{for } r < D_2 \end{cases} \\
V_{\text{swirl-obstacles}} &= \sum_{i=1}^{n} V_{\text{i}}
\end{aligned}
$$

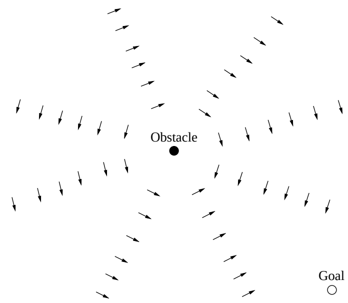

Fig. 4. Example vector field calculated using the Swirl-Obstacles formulae.

A.4 Dock

The Dock motor schema is used to lead the robot around the target and into the appropriate position and orientation for pushing. This behavior is critical to ensure proper alignment of the robot and trailer for our non-holonomic vehicle.

Dock constructs a vector that varies in direction from directly towards the target object to perpendicular for circumnavigation. Outside of a wedge-shaped controlled zone, Dock returns only the perpendicular vector component, while within the controlled zone it returns a linear combination of the two vectors. The formulation of Dock is similar to the docking behavior described by Arkin et al. in [3].

The parameter of this schema is θ_{max} which is used to describe the angular width of the controlled zone (as shown on the left in Figure 5). The right image in Figure 5 shows a sample vector field computed using this approach. The mathematical construction of

Dock is as follows:

$$V_{\mathrm{RtoT}} = \text{vector from center of robot to target}$$
$$V_{\mathrm{perp}} = V_{\mathrm{RtoT}} \text{ +/- 90 degrees depending which way robot should swirl towards target}$$
$$V_{\mathrm{dock}} = \begin{cases} V_{\mathrm{perp}} & \text{for } |\theta_r| > |\theta_{max}| \\ \alpha V_{\mathrm{perp}} + (1-\alpha)V_{\mathrm{RtoT}} & \text{for } |\theta_r| \leq |\theta_{max}| \end{cases}$$
$$\alpha = \frac{\theta_r}{\theta_{max}}$$

A.5 Push

The Push motor schema enables the robot to control the target while moving from its initial position to the goal location. This motor schema first determines the robot's distance from the target and then, based on that distance, constructs a vector to describe the robot's desired heading. If the robot is far away from the target it returns a vector that will place the robot just behind the target, and if the robot is close to the target it returns a vector that places the robot just in front of the target's current position and facing the goal. Alignment is maintained through use of the position of the target relative to the goal. Thus, as the robot gets close to the target it is continually trying to get to a point in front of the target that also gets it closer to the goal location. As the robot drives to this position, it pushes the target in front of it.

The parameters for this schema are D_3, the distance from the target at which this schema switches from calculating a point between the robot and the target to a point between the target and the goal, and λ, the absolute distance from that calculated point to the target. Letting r be the distance from the robot to the target as in the previous sections, mathematically:

$$V_{\mathrm{TtoG}} = \text{vector from center of target to goal}$$
$$V_{\mathrm{RtoT}} = \text{vector from center of robot to target}$$
$$V_{\mathrm{push}} = \begin{cases} V_{\mathrm{RtoT}} - \lambda V_{\mathrm{TtoG}} & \text{for } r > D_3 \\ V_{\mathrm{RtoT}} + \lambda V_{\mathrm{TtoG}} & \text{for } r \leq D_3 \end{cases}$$

B. Behavioral Assemblages

This section describes how the primitive behaviors (motor schemas) are combined into more complex behavioral assemblages capable of solving components of the pushing task. Table I lists the parameters used for each motor schema as well as the gains used to linearly combine those schemas into assemblages.

B.1 Search

The Search assemblage enables the robot to search for the target object. Due to the wide field of view of the robot's camera and the relatively large distance at which it can detect a target, in-place scanning allows the robot to see most of its environment and successfully search for and identify a target.

The Search assemblage was formed using only the Scan motor schema with the scan parameter X set to 90 degrees. This allows the robot to rotate a total of 180 degrees; the maximum allowed by the trailer constraint.

Recall that the movement direction for the robot is determined through combination of the output vectors of active motor schemas, and that the speed of the robot is determined separately. In the Search assemblage there is only one active schema, Scan, and the speed is set to zero. The net result is a scan-in-place behavior.

B.2 Acquire

The Acquire behavioral assemblage positions the robot in a location that enables it to deliver the target to the goal. To accomplish this, the robot must navigate to a specific position and orientation with respect to the target and goal. Our approach is to combine the Dock and Go-To-Target schemas to generate the alignment part of this assemblage. Dock directs the robot around the target to the proper side for pushing while Go-To-Target draws the robot to the target. These two motor schemas are blended such that at a far distance from the target, the robot is using only the Go-To-Target schema and at a close distance only the Dock schema. In between these distances, the outputs of the two schemas are linearly combined. Dock is weighted higher as the robot gets closer to the target.

In addition to the alignment vectors, the Acquire behavioral assemblage also includes the Swirl-Obstacles motor schema. This schema is included to help prevent collisions with obstacles.

The alignment phase of the Acquire behavioral assemblage is parameterized by C_4 and D_4, the radii of the controlled and dead zones, defined similarly to C_1 and D_1 as shown in Figure 3. Mathematically, with r defined as for the Go-To-Target schema:

$$V_{\mathrm{alignment}} = \beta V_{\mathrm{go-to-target}} + (1-\beta)V_{\mathrm{dock}}$$
$$\beta = \begin{cases} 1 & \text{for } r > C_4 \\ \frac{r-D_4}{C_4-D_4} & \text{for } D_4 \leq r \leq C_4 \\ 0 & \text{for } r < D_4 \end{cases}$$

B.3 Deliver

The Deliver behavioral assemblage, activated when the robot has acquired the target object, is used to move this object from its original location to the goal location. This assemblage combines the Push and

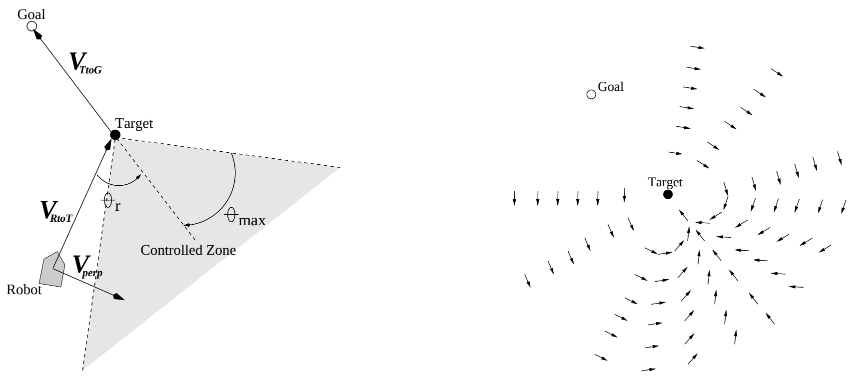

Fig. 5. The Dock motor schema. Left: Parameters used in the calculation of Dock. Right: Example vector field calculated by the Dock schema.

Swirl-Obstacles motor schemas in order to allow the robot to accomplish its task while avoiding obstacles.

TABLE I
Experimental motor schema parameter and gain values.

Behavioral Assemblage	Motor Schemas	Gain
Search	**Scan**	1.0
	X = 90degrees	
Acquire	**Go-To-Target**	0.3$*\beta$
$C_4 = 1.5$m	$C_1 = 0.0$m	
$D_4 = 0.7$m	$D_1 = 0.0$m	
	Dock	0.3$*(1-\beta)$
	θ_{max} = 68degrees	
	Swirl-Obstacles	0.3
	$C_2 = 1.2$m	
	$D_2 = 0.75$m	
Deliver	**Push**	0.3
	$\lambda = 0.223$m	
	$D_3 = 0.8$m	
	Swirl-Obstacles	0.3
	$C_2 = 1.2$m	
	$D_2 = 0.75$m	

IV. Experimental Setups

Two types of pushing tasks were investigated: box pushing and ball dribbling during a soccer game. The generalized control system was first developed and tested in the TeamBots simulator. Once the system was working well in simulation, it was run on mobile robots. Further refinement was required due to small differences between the simulated and physical robots.

The box pushing experiments involved locating an orange box of size 25.5cmx17cmx10cm and pushing it to a 30cmx30cm goal location. In these experiments, five black boxes served as obstacles. The environment in which this task took place was quite noisy with other types of obstacles present and variable lighting conditions. The robot used odometry to locate the goal location's center which was given *a priori*. This setup is illustrated in Figure 6.

The second task is ball dribbling for robotic soccer. The CMUHammerhead team, competing at RoboCup 2000 [7], made use of the generalized control system described here for the team's forwards and a modified version for the halfback and goalie. The RoboCup field measured 5mx9m with two goals each having a width of 2m and a depth of 0.9m. The team members used their cameras to locate the center of the goal on which they should score. Three teammates and four opponents constituted seven moving obstacles that were identified by color. The target was an orange soccer ball (approximately 10cm in radius). This task is more challenging than simple box pushing because the the obstacles are not static and the robot has much less control of the ball than it does over a box. Ball handling sticks were added to the robot to provide more positive control. Even so, the ball often rolled away from the robot and opponents frequently attempted to steal it away.

V. Results

In both tasks calculation of the pushing behavior's output takes about 15ms per control cycle. Additional computational requirements include: 20ms used by the lower level software to translate control system output into appropriate motor velocities, 50ms to communicate those velocities to the drive unit, and about 12ms to handle vision processing. Thus, on average the control system runs at about 10Hz, with the main overhead being serial communication between the computer and the drive unit.

In the box pushing task the robot was able to repeatably navigate to a box and deliver it to the goal location (Figure 6). Once delivered, the robot waited until the box was removed before starting to search for it again. If the box was removed from the robot or

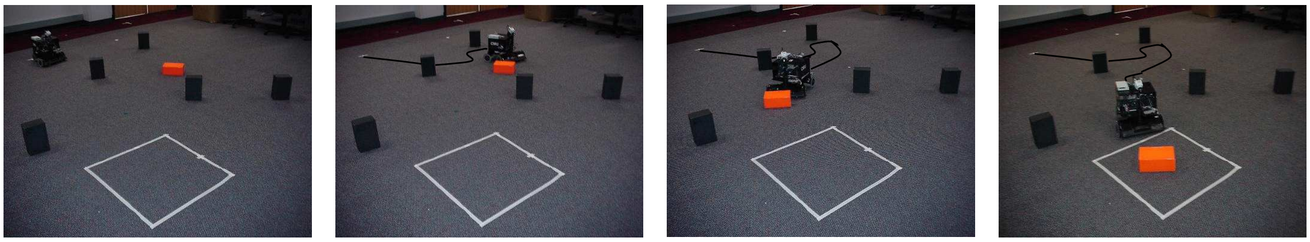

Fig. 6. Results from a box pushing task. Starting at the left picture, the robot acquires the box and pushes it to the goal. Once the box is at the goal the robot backs up. The robot's path is shown by the black line.

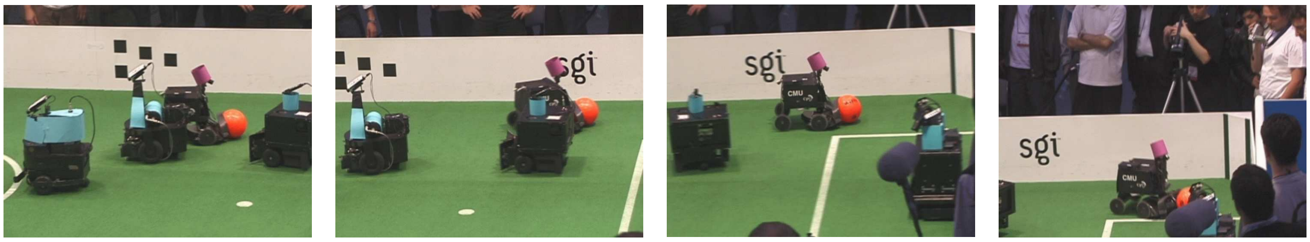

Fig. 7. CMUHammerhead forward dribbling a ball into the goal during a game. As the forward travels towards the goal it maneuvers around its opponents.

it was lost during travel, the robot would successfully recover and re-acquire the box.

To quantitatively evaluate the reliability of the box pushing controller, the task was repeated 30 times. At the beginning of each experimental trial the robot was started at the goal location facing in a random direction. The test environment was about 5m in diameter. For each trial, the target box was placed at a random location between 1.0m and 2.5m from the goal location. Obstacles were placed roughly in a circle around the goal location.

During the evaluation the robot would occasionally travel outside of the arena boundary and interact with other objects in the room. These objects were not painted black so they were not recognized by the vision system as obstacles. The robot, however, was able to recover in the majority of cases by detecting a collision and backing up.

Out of 30 trials, the robot failed to deliver the box to the goal four times; twice because of collisions with outside objects, once because of problems with the longer carpet pile at the edge of its environment, and once because it could not locate the box. Of these failures, only the last one (failure to find the box) indicates a problem with the pushing controller itself. Thus the controller failed only once in 27 trials, or about 4% of the time.

A summary of the results is provided in Table II. The average time taken to complete runs was calculated using the successful 26 runs. During some of the trials the robot bumped one of the black obstacles. This is not considered a failure mode because the robot is able to sense this condition using a bump sensor (based on motor current detectors) and recover. "Re-acquiring the box" is defined as the robot losing the box during the Deliver portion of its behavior and having to go back into Acquire to realign itself.

TABLE II

Results from box pushing trials.

Average time to complete run	50.02s
Standard Deviation	26.12s
Maximum time to complete run	134.44s
Minimum time to complete run	24.07s
Runs box lost and re-acquired	9
Runs bumped black obstacles	10
Runs stuck on carpet	3 (with 1 non-recovery)
Runs bumped outside object	7 (with 2 non-recoveries)
Runs failed to find target	1

The performance of this behavior-based control system in ball dribbling was evaluated during actual robotic soccer matches. At RoboCup 2000, the CMUHammerhead forwards scored a total of nine goals in seven games. In order to score these goals the forwards had to acquire the ball several times and successfully navigate both to the goal and around the goalie. Figure 7 illustrates one of the CMUHammerheads in action.

VI. Limitations

Our approach to box-pushing works quite well in practice. However, because it is based on *local* information only, there is no guarantee of completeness

— it is possible for the robot or the target object to become stuck in a potential well or box canyon. One way to address this would be to integrate a traditional planner with this behavior-based approach; the planner would take over when a lack of progress is detected. This hybrid approach would enable the robot to benefit from the speed of a behavior-based solution most of the time, but still provide guarantees of completeness in more complex environments.

We do not explicitly model the coefficient of friction between the robot and the object it is pushing. Therefore, the robot can make too sharp a turn and lose control of the pushed object. During the Deliver phase, the relationship between the gains for the Push motor schema and the Swirl-Obstacle schema dictate whether the robot is likely to make a sharp turn and lose the target or not. If the Swirl-Obstacle gain is large in proportion to the Push gain, the robot might make a more aggressive turn to avoid an obstacle than is necessary. In practice we set these gains empirically. Fortunately, loss of the target object is not an unrecoverable failure. Because the robot can detect when control of the target object is lost, we can invoke a recovery procedure to recapture it. Loss and recovery of the target occurred in 9 of the 26 successful box-pushing trials.

VII. Conclusions

We describe a behavior-based approach for controlling non-holonomic robots in a pushing task. The controller does not plan a route, but rather calculates an instantaneous heading and speed for the robot based on current sensor readings. The simplicity of the approach enables control commands to be computed quickly (15ms on a 266Mhz Pentium). This control system enables a robot to find a target object, navigate to it, then push the target back to a goal location while simultaneously avoiding obstacle hazards. Non-holonomic constraints of the robot are addressed by a docking behavior that directs the robot around the target and into the correct position and orientation for successful pushing.

The approach is tested on two very different types of targets in static and highly dynamic environments. In box pushing experiments, the system demonstrated 96% reliability in accomplishing the task. The system was also demonstrated at RoboCup-2000, an international robotic soccer competition. At RoboCup, the robots using our approach scored nine goals in seven games.

In the future this work should be extended to determine the extent of the control system's tolerance to differences in friction, shape and size of target objects. This solution to the pushing task will also be used as a building block for future cooperative work in foraging-like tasks.

Acknowledgements

The authors thank Ashley Stroupe and Kevin Sikorski for their contributions during the early development of the pushing behavior and Rande Shern for his construction and maintenance of the Minnow robots. We also thank Jim Bruce, Scott Lenser and Zia Khan for their contributions to the vision system. Finally we thank Manuela Veloso for her ongoing encouragement and advice.

This work was funded by DARPA under the Mobile Autonomous Robot Software Program.

References

[1] P. K. Agarwal, J. C. Latombe, R. Motwani and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. *Proceedings IEEE International Conference on Robotics and Automation*, Albuquerque, NM, 1997.

[2] R. Arkin. Motor schema-based mobile robot navigation *International Journal of Robotics Research*, 8(4):92–112, 1989.

[3] R. Arkin, R. Murphy, M. Pearson and D.Vaughn. Mobile robot docking operations in a manufacturing environment: progress in visual perceptual strategies. *Proceedings IEEE International Workshop on Intelligent Robots and Systems (IROS '89)*, Tsukuba, Japan, pp. 147–154, 1989.

[4] R. Arkin and D. MacKenzie. Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 10(33):276–286, June 1994.

[5] T. Balch. Clay: Integrating motor schemas and reinforcement learning. College of Computing Tech Report GIT-CC-97-11, Georgia Institute of Technology, Atlanta, Georgia, March 1997.

[6] T. Balch. Behavioral Diversity in Learning Robot Teams. Ph.D. Thesis, College of Computing, Georgia Institute of Technology, December, 1998.

[7] T. Balch, P. Stone and G. Kraetzschmar, Eds. *Proceedings of the 4th International Workshop on RoboCup*, Melbourne, Australia, 2000.

[8] J. Bruce, T. Balch and M. Veloso. Fast and cheap color vision on commodity hardware. *Workshop on Interactive Robotics and Entertainment*, April 2000, Pittsburgh PA, pp. 11–15.

[9] J. Cameron. Modeling and Motion Planning for Non-Holonomic Systems (Boltzmann Hamel Equations). Ph.D. Thesis, Georgia Institute of Technology, 1993.

[10] J. C. Latombe. *Robot Motion Planning*, Kluwer, Dordrecht, The Netherlands, 1991.

[11] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability and planning. *International Journal of Robotics Research*, 15(6): 533–556, December 1996.

[12] M. Mataric, M. Nilsson and K. Simsarian. Cooperative multi-robot box-pushing. *Proceedings IROS-1995*, Pittsburgh, PA, 1995.

[13] L. E. Parker. Heterogeneous Multi-Robot Cooperation. PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, February 1994. MIT-AI-TR 1465 (1994).

[14] S. Singh and H. Cannon. Multi-resolution planning for earthmoving. *Proceedings IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 1998.