2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

A Dipole Field for Object Delivery by Pushing on a Flat Surface

Takeo Igarashi, Youichi Kamiyama, Masahiko Inami

***Abstract*—This paper introduces a simple algorithm for non-prehensile object transportation by a pushing robot on a flat surface. We assume that the global position and orientation of the robot and objects are known. The system computes a dipole field around the object and moves the robot along the field. This simple algorithm resolves many subtle issues in implementing reliable pushing behaviors, such as collision avoidance, error recovery, and multi-robot coordination. We verify the effectiveness of the algorithm via several experiments with varying robot and object form factors. Although object delivery by pushing and motion control by a vector field are not new, the proposed algorithm offers easier implementation with fewer parameter adjustments because of its mode-less definition and scale-invariant formulation.**

I. Introduction

OBJECT transportation (or delivery) is one of the most important tasks of a mobile robot. A robot that can automatically deliver an object from one place to another can save human time and labor. The particular class of delivery tasks we are concerned with is non-prehensile delivery by a pushing robot on a flat surface (Figure 1). Most delivery robots use hands or other mechanisms to lift an object, but many important tasks can be achieved by simple pushing.

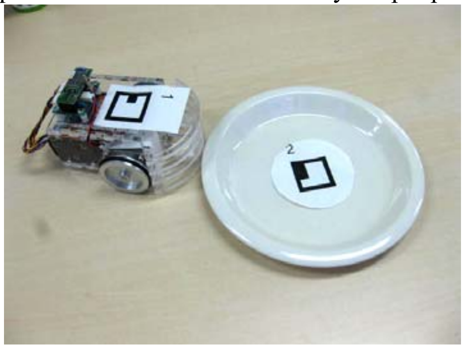
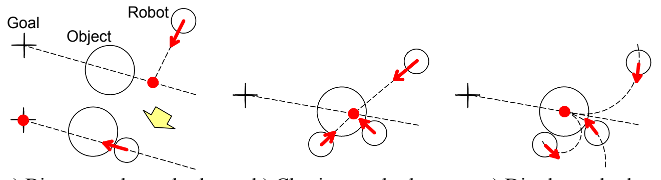


Fig. 1: Non-prehensile delivery by a single robot.

The implementation of an efficient pushing behavior for a single robot is not trivial. The system must compute the global path from the object to the goal, avoiding all obstacles and continuously updating the robot position and orientation, so that the object follows the path during execution. In this work, we focus on the problem of local control, which in itself is difficult to perform efficiently.

We conducted a preliminary study in which computer science students implemented a pushing behavior in a simple simulator world. The task is to push an object to a goal position without considering other obstacles. Most of them chose the naïve approach of moving the robot to the canonical pushing position behind the object and then moving the robot toward the goal (Figure 2a). If the object swerved off course, the robot stopped pushing and returned to the pushing position. However, this binary mode technique is not very efficient, because the explicit recovery operation creates a large overhead. It is also tedious to appropriately set the offset distance and mode switch threshold. Some students used a more sophisticated single mode method (we call this chasing), in which the robot continuously tried to reach a targeted position inside the object and slightly behind its center (Figure 2b). The target position moves as the object moves. This technique averts explicit recovery operations for small perturbations, but the user must carefully set the offset distance, taking into account both robot and object size.



a) Binary mode method b) Chasing method c) Dipole method
Fig. 2: Various pushing algorithms.

This paper introduces a very simple yet robust algorithm for solving the problem of local control for pushing behavior. The objective is to introduce a graceful transition between pushing and recovery to mitigate the issues incurred by the modal approaches. Our algorithm computes a continuous flow field around the object, creating a smooth transition between pushing motion at the desired position and return to this position from other locations (Figure 2c). It is interesting to note that the combination of a robot with a circular cross-section and a simple dipole field solves the problem. The technique is based on a very elegant mathematical definition (see Section III) and is independent of both object and robot size, making the implementation very simple. In this paper, we discuss the definition, derivation, and extensions of this algorithm and provide empirical validation using actual robots.

In a sense, object delivery by pushing is already a solved problem, because successful delivery has been attained in a number of previous experiments in more adverse environments [1][2]. Furthermore, some experiments have used vector fields for robot control, as does our method [3]. Our contribution is in facilitating the implementation of

Manuscript received September 15, 2009.
All authors are with JST ERATO Igarashi Design Interface Project, Tokyo, Japan. Takeo Igarashi is with The University of Tokyo, Tokyo, Japan (e-mail: takeo@acm.org). Masahiko Inami is with Keio University, Tokyo, Japan.

pushing in noisy environments by providing a scale-invariant single vector field representation. We believe that this type of contribution (making things easier rather than making the impossible possible) will become increasingly important as more people join in the development of robot applications.

II. Related Work

Object transportation by pushing has been investigated in connection with parts delivery in a factory [1][2]. In this context, a controlled environment is usually assumed, and the exact location and pushing direction are computed for a specific target object. In contrast, we seek a single general algorithm that works reasonably well for a variety of objects without parameter adjustment. In the parts delivery application, it is also assumed that the robot follows the calculated path perfectly, whereas our algorithm seamlessly integrates pushing and error recovery for an imperfect control system.

Pushing has also been studied extensively in the context of multi-robot coordination [4][5][6][7][8][9]. In these applications, the objective is to implement the coordinated transportation of an object by multiple autonomous robots that communicate with one another without using a global sensor.

Our application of interest is object delivery in a home environment, such as delivering a dish to a table or moving a trash bin from one place to another [10]. Rudimentary pushing behaviors have already been reported in some applied systems [11], but these use a simple binary mode approach (personal communication).

Our algorithm computes a vector field from a given environmental configuration and moves the robot along the field. Many successful examples of this approach have been reported [12][13]. Munasinghe et al. presented an obstacle avoidance using dipole field [16]. The research most closely related to our work was performed by Emery and Balch [3], who developed a control system for a pushing task by a non-holonomic robot. Their technique uses separate modes for dock and push. Dock guides the robot behind the object, using a vector field similar to our dipole, and push moves the robot toward a point with a given offset from the object center (a variant of chasing). Our method unifies the dock and push behaviors into a single scale-invariant formulation, which reduces the need for parameter tweaking when applying it to a new situation.

III. The Algorithm

A. Problem Definition

We introduce a local control algorithm for robot delivery of an object to a goal position on a flat surface by non-prehensile pushing. The algorithm is designed mainly for a circular holonomic robot pushing a circular object, but it is also applicable to a non-holonomic robot with a circular bumper that can spin in place (e.g. a differential drive robot). We also show that the algorithm works reasonably well for non-circular objects (Section V). We assume that the global position and orientation of the robot and object are known to the system via some type of tracking system. We also assume that the global path from the current object position to the goal is specified and that the robot is roughly behind the object. (These problems can be solved separately using any mobile path planning method [14] or object avoidance method [3].)

The input to the algorithm is the object's instantaneous direction of motion (i.e. a tangent to the global path) and the current object and robot location. The output of the algorithm is the robot's desired direction of motion (Figure 3). The system then rotates the robot in the proper direction (in the case of a non-holonomic robot) and moves the robot a pre-defined distance. Some parameter adjustment (e.g. motor torque and timing) is necessary to physically maneuver the robot, but the computation of the robot's desired direction of motion can be handled independently of these low-level control issues.

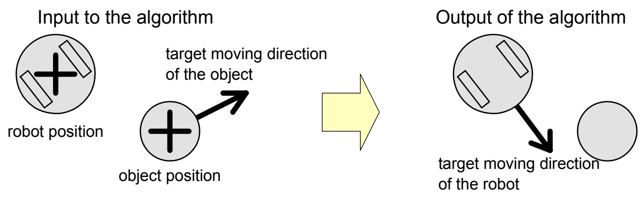
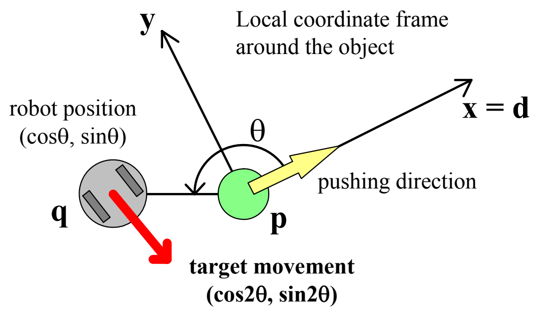


Fig. 3: Problem setting.

B. The Algorithm

The system first defines a local coordinate frame whose origin is at the center of the object, with x-axis parallel to its desired direction of motion. The system then computes the polar coordinates (r, θ) of the robot in this local coordinate frame. The desired direction of motion of the robot in this local coordinate frame is given by (cos2θ, sin2θ). Note that this definition is scale invariant. It is independent of the robot–object distance and the sizes of the robot and the object. Figure 4 illustrates the algorithm and a pseudo-code for it.


y
Local coordinate frame around the object
robot position (cosθ, sinθ)
θ
x = d
pushing direction
q
p
target movement (cos2θ, sin2θ)

```
compute(d = desired direction of motion,
             p = object position, q = robot position){
   x = d; y=rotate(x, $\pi/2$);
   $\theta$ = compute_angle(x, q-p);
   return  normalize(x $cos2\theta$ + y $sin2\theta$);
}
```

Fig. 4: The algorithm.

C. Derivation and Analysis

The above definition is derived by carefully blending two different behaviors: orbiting and pushing (Figure 5). When the robot is directly behind the object (i.e. –cosθ is large), the robot should move toward the object to push it. The pushing direction is given by (–cosθ, –sinθ) in the local coordinate frame. If the robot is not in the correct position behind the object (i.e. sinθ is large), the robot should move toward the correct position by orbiting around the object to avoid collision. The orbiting direction is given by (–sinθ, cosθ). We then blend these directions together with the corresponding weights, yielding the above definition.

$$\underbrace{-\cos\theta}_{\text{weight}}\underbrace{\begin{bmatrix}-\cos\theta\\-\sin\theta\end{bmatrix}}_{\text{push}}+\underbrace{\sin\theta}_{\text{weight}}\underbrace{\begin{bmatrix}-\sin\theta\\\cos\theta\end{bmatrix}}_{\text{orbit}}=\begin{bmatrix}\cos 2\theta\\\sin 2\theta\end{bmatrix}$$

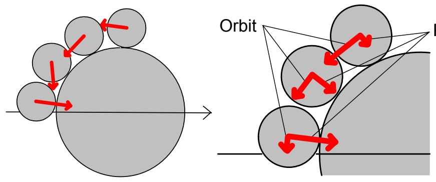


Fig. 5: Derivation of the algorithm.

It is difficult to prove that this specific formulation is the best possible definition. However, we have tested numerous variations and have empirically confirmed the effectiveness of our definition (see Section VI). Moreover, we feel that its simplicity and elegance have a certain appeal. It is interesting to note that the derived vector field (cos2θ, sin2θ) resembles a dipole in physics. A dipole occurs when positive and negative electric charges or opposite electric currents exist with a small separation between them (Figure 6). Geometrically, this corresponds to circular trajectories that are tangent to a point on a line.

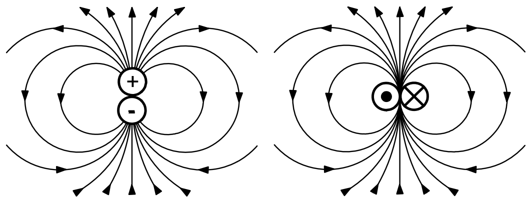


Fig. 6: Dipoles in physics.

IV. Extensions

A. Escaping from the Front Side

One problem with the basic dipole method is that the path from the front to the back of the object is too long and therefore inefficient. In the extreme case when the robot is right in front of the object, the robot moves infinitely far away from the object. (This happens when the robot pushes the object past the goal.) There are many methods of augmenting the basic algorithm to solve this problem, but we have found that the following simple strategy works well in practice.

If the robot is closer to the goal position than the object, the robot simply orbits the object at a constant radius until it reaches the backside. This can be accomplished by simply nullifying the radial component of the desired direction of motion (**v**), as computed by the dipole algorithm, if **v** is pointing away from the object (if the robot is directly in front of the object, the robot moves in a random direction). In a pseudo-code, we replace the last line of the basic code (Figure 4) with the following four lines:

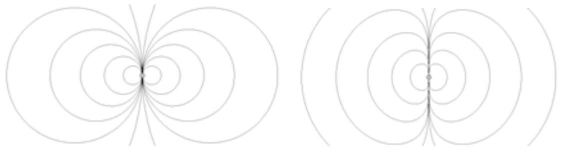
```
v = x cos2θ + y sin2θ;
if (cos(v, n = (p-q)/|p-q|) < 0)
    v = v – (n•v)n;
return normalize(v);
```

In addition to being easy to understand and simple to implement, this method has the following advantages. First, the resulting vector field is smoothly connected to the dipole field at the transition. This stabilizes the resulting robot motion near the boundary. (The robot can repeatedly change direction if the flow direction is discontinuous as in a naïve binary controller.) Second, this algorithm is scale independent. One can use the same code regardless of object and robot size. This means that there is no need for parameter tuning, which greatly reduces the labor involved in practical deployment.

B. Avoiding Excessive Orbiting near a Small Goal

The basic dipole method works well when the object is far away from the goal position or the goal is larger than the object, but it does not work well when the object is approaching a relatively small goal. In the worst case scenario, the robot and object start orbiting endlessly around the goal. This occurs because the dipole method pushes the object sideways when the robot tries to move to the back of it.

To solve this problem, we propose increasing the orbiting factor when the object is close to the target. This method can be regarded as a generalization of the original method and one that preserves its scale-independent nature. The robot's desired direction of motion is given by ($\cos^2\theta-\alpha\sin^2\theta$, $\sin\theta\cos\theta+\alpha\sin\theta\cos\theta$), where α is a weighting factor. Larger values of α result in slower but more cautious pushing behavior. α = 1 is the default dipole motion (Figure 7).


a) α = 1 b) α = 4
Fig. 7: Weighting the orbiting factor.

We increase the weighting factor α as the object gets closer to the goal position. Specifically, we define $\alpha = |\theta/\phi|$, where ϕ is the angle between the robot–object and robot–goal vectors and θ is the angle between the object–robot and goal–object vectors (Figure 8). The idea is to make α inversely proportional to ϕ and to make it larger when the distance between the object and the goal is small relative to the distance between the robot and the object. We then multiply by θ to prevent α from being infinitely large when the robot, object, and goal are almost aligned. Figure 9 shows the resulting vector field, combining the extensions described in Sections IV-A and -B. In practice, excessively large values of α cause abrupt turns behind the object, making it difficult to control. We therefore set an upper limit for α (10 in our experiments).

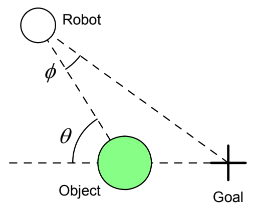


Fig. 8: Angles used to compute the weight α.

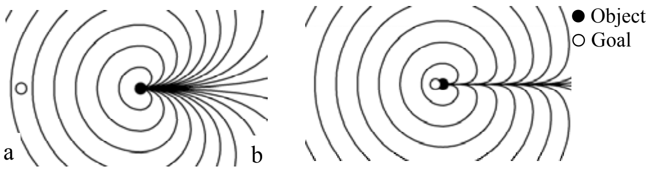


Fig. 9: Final vector field. a) Goal is far away to the left. b) Goal is nearby.

C. Pushing a Heavy Object with Multiple Robots

An object might be too heavy for a robot to push, necessitating the assistance of an additional robot. Our algorithm can address this simply by setting the dipole field for guiding the second robot around the first robot so that its orientation parallels the orientation of the first robot. The extension described in the previous subsection requires the distance between the goal and the object. For the current purpose, we use the distance between the object and the first robot. The second robot is made to push the first robot from behind.

Typical multi-robot pushing methods place the robots in parallel around the object [4]. However, parallel pushing is difficult when the target object is smaller than the robot, and our serial pushing method can be useful in such cases. However, parallel pushing is advantageous when the object is larger than the robot, and we plan to investigate techniques that support parallel pushing in future work.

V. IMPLEMENTATION

A. Hardware

We implemented a prototype system to demonstrate the feasibility of the proposed method. This system consists of a mobile robot, a ceiling-mounted camera, and a host computer (Figure 10). The host computer uses the camera to track the position and orientation of the robot and object and wirelessly sends control signals to the robot. We used a notebook PC (Toshiba Dynabook SS RX1 with an Intel Core™ 2 Duo ULV U7500 1.06 GHz processor running Windows XP) and a USB web camera (Logicool Qcam Pro for Notebook, 2M pixels).

We tested two types of non-holonomic robots in different sizes to illustrate the versatility of the algorithm. One was a small custom-made differential drive tabletop robot with a circular bumper running. The other was a commercially available robot running on a carpeted floor (iRobot Create).

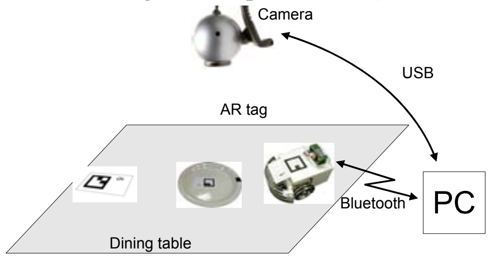


Fig. 10: Hardware configuration.

B. Software

The tracking and control program was written in Java™. It uses a simple 2D tracking system with proprietary visual markers, similar to the AR toolkit [15]. The markers display a 3 × 3 black-and-white pattern enclosed in a black frame, enabling the system to uniquely detect identity, position, and orientation. It uses 5× 5 cm markers when the camera is 1 m above the surface, and 10 × 10 cm markers when the camera is 2 m above the surface. The camera resolution is 640 × 480 at 30 fps, with a delay of approximately 30 ms.

The control program receives the updated marker position and orientation and sends control commands to the robot. It first computes the object's desired instantaneous direction of motion as a tangent of the global path toward the goal. It then computes the robot's desired direction of motion using the algorithm described in the previous section. Finally, it sends low-level control commands (move forward, spin left, spin right, and stop) to the robot in accordance with the computed direction of motion.

There are many ways of driving a non-holonomic robot in a given direction, but a very basic method is used in this study. The system compares the difference between the current robot orientation and the desired direction of motion. If the difference is less than a small threshold value, the robot starts to move forward (or continues moving). Otherwise, the robot spins (or continues spinning).

The system repeats the above procedure as necessary and terminates the process when the distance between the object and the goal is smaller than a chosen tolerance.

VI. EMPIRICAL EVALUATION

To verify the effectiveness of the algorithm, we ran two experiments using the two types of robots. Figure 11 illustrates the layout. A trial was considered to be complete when the distance between the goal position and the object center was less than 20 pixels in the camera view (40 pixels in the case of the iRobot Create). A trial was considered to be a failure if the robot or the object went outside the field and/or the system was in an infinite loop (in which case we terminated the execution after 5 min).

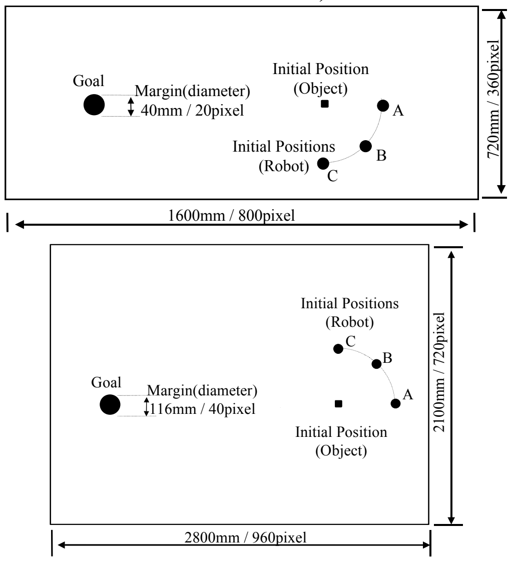


Fig. 11: Layout. Top: Tabletop robot. Bottom: iRobot Create.

A. *Scale Invariability*

The goal of the first experiment was to show that the proposed algorithm works well for various configurations without changing parameters. We tested two circular objects of differing sizes, and a square object with bumpers of two different sizes attached to the tabletop robot. We also tested two boxes of differing sizes with the iRobot Create. Figure 12 displays the measurements. We tested two control algorithms, one being the proposed dipole method (with the two extensions described in Sections IV-A and -B) and the other being the simple chasing method. With the chasing method, we tested two different offset values (distance between the object center and the target position of the robot), one being manually adjusted to work well for a small object and the other being adjusted for a large object. We ran 10 trials for each setting.

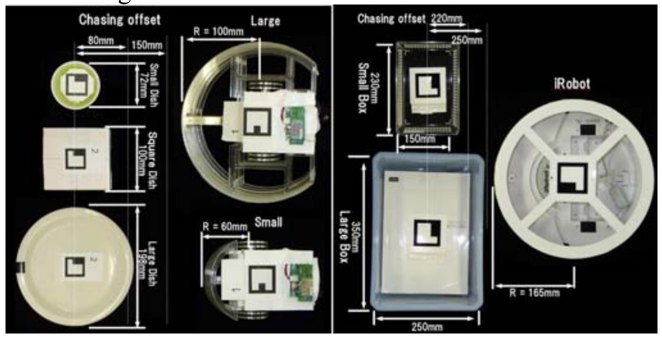


Fig. 12: Robot and object sizes.
Left: Small tabletop robot. Right: iRobot Create.

Tables 1 and 2 contain the results, and Figure 13 shows sample trajectories observed during the experiment. Our algorithm successfully delivered objects of varying sizes and shapes to a small goal position without changing the parameters. In contrast, a single parameter setting does not work well under different conditions when using the chasing algorithm (as shown in the failures cases in Table 1).

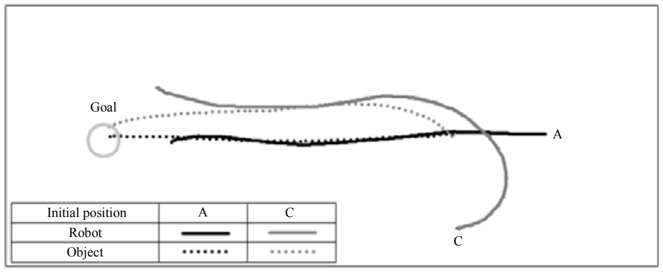


Fig. 13: Typical object and robot trajectories in the tabletop setting.

Table 1: The result of pushing by the tabletop robot
(success ratio out of 10 trials)

dish size	initial position	dipole		chasing (d = 80mm)		chasing (d = 150mm)	
		small bumper	large bumper	small bumper	large bumper	small bumper	large bumper
small dish	A	100	100	100	50	0	0
	B	100	100	100	80	0	0
	C	100	100	100	90	0	0
large dish	A	100	100	60	10	100	100
	B	100	100	70	0	100	100
	C	100	100	0	0	100	10
square dish	A	100	100	100	30	0	0
	B	100	100	100	100	0	0
	C	100	100	100	0	0	0

Table 2: The result of pushing by iRobot Create
(success ratio out of 10 trials)

box size	initial position	dipole	chasing (d=220mm)	chasing (d=250mm)
small box	A	100	100	0
	B	100	100	0
	C	100	100	0
large box	A	100	60	100
	B	100	70	100
	C	100	0	100

B. *Pushing with Multiple Robots*

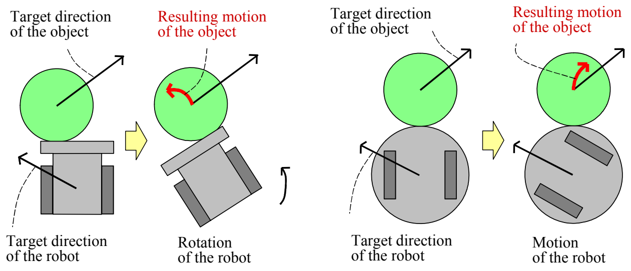
The second experiment was designed to demonstrate that the proposed algorithm can be used to control multiple robots pushing a heavy object. The iRobot Create was used in this experiment. Two different boxes (3.5 or 7 kg) were pushed by a single robot and by two robots controlled by the algorithm described in Section IV-C. We ran 10 trials in each setting. Table 3 contains the results, which show that two robots can successfully deliver an object that is too heavy for one robot to move. However, three robots do not improve success ratio over two, probably because of overhead resulting from mechanical interference (friction, etc.).

Table 3: The result of pushing by multiple robots
(success ratio out of 10 trials)

# of robots	initial positions	light box (3.5kg)	heavy box (7kg)
single robot	A	100	0
	B	100	0
	C	100	0
two robots	A,B	100	100
	A,C	100	100
	B,C	100	100
	C,A	100	100
	C,B	100	100
	B,A	100	100

VII. Discussion

An interesting fact that emerges from this study is that a circular front bumper works better than a flat bumper for pushing. At first we naïvely anticipated that a flat bumper would be the better choice and so tested it. It turns out that the flat bumper works well as long as the object remains in front of the robot. However, the object inevitably swerves away, and the robot must adjust its position accordingly. As soon as the robot starts spinning to adjust its position, the bumper pushes the object in the wrong direction, eventually losing contact. Thus, the robot must retreat from the object and then approach it from the correct direction, which creates significant overhead. In contrast, when the robot has a circular bumper, the object is not pushed during spinning. The object is then pushed in the desired direction when the robot returns to the correct pushing position in accordance with our algorithm. Figure 14 illustrates this. It is possible to attach an arm to hold the object, but a circular bumper is advantageous because it can handle a large object without having a large arm.


Target direction of the object
Resulting motion of the object
Target direction of the object
Resulting motion of the object
Target direction of the robot
Rotation of the robot
Target direction of the robot
Motion of the robot
Fig. 14: Flat vs. circular bumper. The flat bumper pushes the object in the wrong direction during spinning (left), whereas the circular bumper pushes the object in the correct direction (right).

VIII. Conclusion and Future Work

We developed an algorithm for object delivery by an armless robot pushing on a flat surface. The algorithm has a very simple definition in terms of (cos2θ, sin2θ), similar to a dipole in physics. We also introduced extensions to resolve practical issues when applying the basic concept to real-world problems. We empirically validated its effectiveness by testing it with various configurations. Pushing is a very basic operation, and we believe that the proposed algorithm has practical value.

Our immediate future objective is to develop an algorithm for a four-wheeled car with front-wheel steering. The algorithm presented here assumes in-place spinning and is not designed to handle such a car. It might be possible to continuously push the object when it swerves away, as in the current algorithm, but it might first be necessary to back in order to avoid contact with the object.

The current algorithm ignores dynamics (inertia), assuming relatively slow robot motion and high friction. However, dynamics must be taken into account to support faster robot motion and/or a low friction surface. Faster robots are a challenge, because they need accurate control and tracking mechanisms, but high-speed object delivery by pushing is certainly an interesting research goal.

References

[1] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. International Journal of Robotics Research, 15(6): 533–556, December 1996.
[2] K. M. Lynch. Locally controllable manipulation by stable pushing. IEEE Transactions on Robotics and Automation, 15(2):318–327, April 1999.
[3] Emery, R. and Balch, T., Behavior-Based Control of a Non-Holonomic Robot in Pushing Tasks, IEEE International Conference on Robotics and Automation (ICRA-2001), Seoul, 2001.
[4] M. J. Mataric, M. Nilsson, and K. Simsarian, Cooperative multi-robot box-pushing, in Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Pittsburgh, Pennsylvania, August 1995, pp. 556–561.
[5] D. Rus. Coordinated manipulation of objects in the plane. Algorithmica, pages 129–147, 1997.
[6] Fink, J., Hsieh, M.A., Kumar, V., Multi-robot manipulation via caging in environments with obstacles, ICRA 2008. IEEE International Conference on Robotics and Automation, 2008, 19–23 May 2008 Page(s):1471 – 1476
[7] C. R. Kube and H. Zhang. Collective robotics: from social insects to robots. Adaptive Behaviour, 2(2):189{218, 1993.
[8] C. Ronald Kube and Eric Bonabeau. Cooperative transport by ants and robots, Robotics and Autonomous Systems, Vol. 30, Issue 1–2, pages 85–101, 2000.
[9] B. P. Gerkey and M. J. Mataric´, "Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination," in Proc. IEEE Int. Conf. Robotics and Automation (ICRA), Washington, DC, May 2002, pp. 464–469
[10] X. Xu, T. Deyle, C. Kemp, "1000 Trials: An Empirically Validated End Effector That Robustly Grasps Objects from the Floor", 2009 IEEE International Conference on Robotics and Automation, May 12–17, 2009, Kobe, Japan.
[11] S. Zhao, K. Nakamura, K. Ishii, and T. Igarashi, "Magic Cards: A Paper Tag Interface for Implicit Robot Control", Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI2009, pp. 173–182, Boston, USA, April, 2009.
[12] A. Masoud, "Harmonic Potential Field Approach with a Probabilistic Space Descriptor for Planning in Non-Divisible Environments", 2009 IEEE International Conference on Robotics and Automation, May 12–17, 2009, Kobe, Japan
[13] R. C. Arkin, Behavior-Based Robotics. New York, NY: Cambridge University Press, 1998.
[14] Hwang, Y. K. and Ahuja, N. 1992. Gross motion planning—a survey. ACM Comput. Surv. 24, 3 (Sep. 1992), 219–291.
[15] H. Kato, M. Billinghurst, B. Blanding, and R. May, "ARToolKit". Technical Report. Hiroshima City University. December 1999.
[16] Munasinghe, S.R., Oh, C., Lee,J.J. and Khatib, O., "Obstacle avoidance using velocity dipole field method", International Conference on Control, Automation, and Systems, ICCAS 2005, in Kintex, Gyeong Gi, Korea,1657 - 1661.