NEO: A Novel Expeditious Optimisation Algorithm for Reactive Motion Control of Manipulators

Jesse Haviland and Peter Corke

Abstract—We present NEO a fast and purely reactive motion controller for manipulators which can avoid static and dynamic obstacles while moving to the desired end-effector pose. Additionally our controller maximises the manipulability of the robot during the trajectory while avoiding joint position and velocity limits. NEO is wrapped into a strictly convex quadratic programme which when considering obstacles joint limits and manipulability on a 7 -of-freedom robot is generally solved in a few ms. While NEO is not intended to replace state-of-the-art motion planners our experiments show that it is a viable alternative for scenes with moderate complexity while also being capable of reactive control. For more complex scenes NEO is better suited as a reactive local controller in conjunction with a global motion planner. We compare NEO to motion planners on a standard benchmark in simulation and additionally illustrate and verify its operation on a physical robot in a dynamic environment. We provide an open-source library which implements our controller.

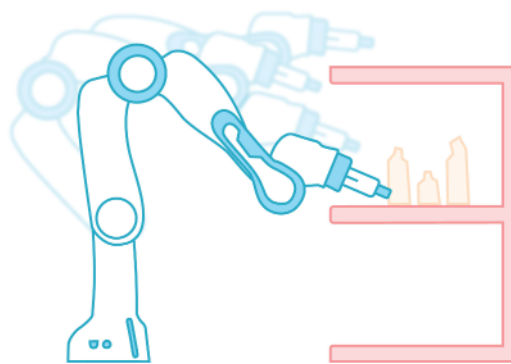*Index Terms*—Motion control collision avoidance and reactive control.

Fig. 1. Our NEO controller drives a robot's joint velocities such that the robot does not collide with obstacles (that may be non-stationary), does not exceed joint position (translational or angular) or velocity limits and maximises the manipulability of the robot, purely reactively.

I. INTRODUCTION

THE real world is not static and consequently, manipulators must be up to the challenge of reacting to the unpredictable nature of the environment they work in. Getting a manipulator's end-effector from point A to point B is a fundamental problem in robotic control. While the task is seemingly simple, there can be numerous causes of failure, especially as the environment becomes cluttered and dynamic. Dynamic obstacle avoidance is essential for manipulators to be robust in such environments.

Each iteration of the control loop must be able to consider the state of both environment and robot to guarantee safe and reliable robot operation. The current focus in obstacle avoidance for manipulators has come at the cost of greater up-front computational load, leading to open-loop motion plans. Planners compute a sequence of joint coordinates which at run time become joint velocities. In contrast, a reactive approach capable of avoiding non-stationary obstacles works directly with joint velocities.

We consider the differential kinematics and compute a set of joint velocities that *steer* the end-effector towards the goal in task space. This classic and purely reactive approach is known as resolved-rate motion control (RRMC) and is cheap to compute and easily able to run at over 1000 Hz. In this letter we add additional capabilities to the RRMC approach while maintaining its highly reactive capability.

Differential kinematics also allows us to capture the rate of change of the distance between any part of the robot and any obstacle present in the environment. By exploiting such relationships, a reactive controller can be developed that will avoid colliding with obstacles. The resulting controller may be over-constrained and unable to achieve the goal pose, but we can employ two strategies to resolve this.

Firstly, we could employ a kinematically redundant manipulator with more degrees of freedom than is necessary to reach any pose within its task space – these are now increasingly common. Secondly, we could relax the path specification to allow for intentional error, called *slack* which allows the end-effector to diverge from the straight-line trajectory to dodge obstacles.

To provide the best chance for the robot to be able to react in a volatile environment, we must consider the robot's conditioning. A measure of manipulability, devised in [1], describes how well-conditioned a manipulator is to achieve any arbitrary velocity. Therefore, by also maximising the manipulability of the robot, we can decrease the likelihood of robot failure due to singularity while also improving robot obstacle avoidance.

The contributions of this letter are:

1) a reactive motion controller for serial-link manipulators (fully actuated or redundant) that can achieve a desired end-effector pose with the ability to dodge stationary and non-stationary obstacles, avoid joint limits and maximise manipulability.
2) experimental validation in simulation on a published motion planning benchmark as well as on a physical Franka-Emika Panda robot.
3) an open-source Python library providing all tools necessary to implements our controller on any serial-link manipulator [2]. Benchmark code and implementation details are provided at jhavl.github.io/neo.

II. RELATED WORK

Classical planning methods will find the optimal path according to the supplied criteria. However, their computation time means they are restricted to use as open-loop planners. Motion planners such as STOMP [3], CHOMP [4] and Trajopt [5] use sampling-based methods to solve general high-dimensional motion planning problems not just for manipulators. Motion planners can typically incorporate collision avoidance as well as other constraints such as end-effector orientation throughout the motion. In some cases, motion-planners may need post-processing to remove jerky or extraneous motions due to sampling.

STOMP uses a stochastic and gradient-free scheme to optimise a trajectory while minimising a cost function. The stochastic nature of the scheme means that it can overcome local minima which purely gradient-based approaches like CHOMP and Trajopt can be trapped by. CHOMP and Trajopt both utilise novel methods for solving optimisation problems while incorporating obstacles.

Planning times of these algorithms preclude their use in dynamic environments. In one benchmark [5] planning time varies from 0.19 seconds (for Trajopt) to 4.91 seconds (for CHOMP).

Path planners output a sequence of poses for which the inverse kinematic (IK) solution is computed to obtain a sequence of joint coordinates. IK can be solved analytically on most robots, but it is more commonly solved through an optimisation framework [6]. The optimisation problem can be augmented, through the cost function, with equality or inequality constraints to provide additional functionality. IK solvers can therefore provide joint configurations which are free from collisions and avoid joint limits. Typically only redundant robots with an exploitable null space can achieve useful benefits.

However, IK alone cannot provide a valid path to the goal. Rather than consider the pose or joint configuration at each time step we can consider velocity. This approach, using differential kinematics, is commonly referred to as Resolved-Rate Motion Control (RRMC) [7].

RRMC uses the derivative of the forward kinematics to choose joint velocities which will drive the robot in a straight line from the starting end-effector pose to the desired end-effector pose. Unlike the previously mentioned approaches, RRMC is purely reactive and provides the joint velocities for the next time instant rather than for the whole trajectory.

If the robot is redundant, there are an infinite set of joint velocities which will achieve the desired end-effector velocity. Typically an additional constraint such as minimising the norm of joint velocity is applied. Alternatively, through gradient projection and exploitation of the null space, the robot can complete sub-tasks while achieving its motion.

One such subtask is maximising the manipulability of the robot [8]. By projecting the manipulability Jacobian into the null space of the differential kinematic equation, the manipulator will perform the straight-line end-effector motion towards the goal pose, but also choose joint velocities such that the robot manipulability is maximised at each step. This is beneficial for the robustness of the robot operation as it decreases the likelihood the robot will approach a singular configuration. While there are numerous measures for kinematic sensitivity, the Yoshikawa manipulability index [1] is the most widely accepted and used measure for kinematic manipulability. The manipulator Jacobian contains rows relating to translational and rotational velocities. Additionally, manipulators comprising mixed prismatic and revolute actuators have non-homogeneous Jacobians due to the different units used. Therefore, when using Jacobian-based manipulability measures such as [1] care must be taken to consider issues including length scaling and mixed units [9]. In a dynamic environment, manipulability optimisation enables the robot to achieve arbitrary velocity at the next time step in order to meet the needs of unpredictable scene dynamics.

RRMC can be reformulated as a quadratic programming (QP) optimisation problem. QPs provide the ability to consider constraints and costs which provide additional functionality. The work in [10] incorporated the physical joint limits of a mobile manipulator into a QP. Recent work in [11] introduced slack into the typical straight-line approach of RRMC – effectively introducing extra redundancy to the problem which is especially useful for robots with six or fewer degrees-of-freedom. This work then exploited the extra redundancy to maximise the manipulability and avoid physical joint limits of manipulators through a reactive QP controller. The velocity damper approach described in [12] was used to avoid joint limits but could equally be used to avoid obstacles.

A classical and alternate approach to reactive motion control is through potential fields where the robot is repelled from joint limits and obstacles [13], [14]. This guides the tool on a collision-free path to the goal but does not explicitly maximise the manipulability or conditioning of the robot.

Reactive control is essential for important sensor-based motion robotics problems such as visual servoing [15] or closed-loop visual grasping [16]. Both determine the end-effector velocity for the next control loop iteration, and this is challenging to achieve with planning-based methods.

This letter proposes a novel real-time motion controller which can avoid moving obstacles (for the whole robot, not just the end-effector), avoid physical joint limits and maximises the manipulability of a manipulator in a purely reactive manner.

In Section III we incorporate the differential kinematics and manipulability maximisation into a general QP problem. We extend this to handle obstacles in Section IV, then velocity dampers and slack variables in Section V. We present the NEO

controller in Section VI and our experimental methodology in Section VII. We conclude in Section VIII with a discussion of our experimental results and insights.

III. QUADRATIC PROGRAMMING

The generic form of a QP is [17]

$$\min_{x} \quad f_o(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x},$$

$$\text{subject to} \quad \boldsymbol{A}_1\boldsymbol{x} = \boldsymbol{b}_1,$$
$$\boldsymbol{A}_2\boldsymbol{x} \leq \boldsymbol{b}_2,$$
$$\boldsymbol{d} \leq \boldsymbol{x} \leq \boldsymbol{e}. \tag{1}$$

where f_o is the objective function to minimise; $\boldsymbol{A}_1, \boldsymbol{b}_1$ define the equality constraints; $\boldsymbol{A}_2, \boldsymbol{b}_2$ define the inequality constraints, and $\boldsymbol{d}$ and $\boldsymbol{e}$ define the lower and upper bounds of the optimisation variable $\boldsymbol{x}$. When the matrix $\boldsymbol{Q}$ is positive definite the QP is strictly convex [17].

A. Incorporating the Differential Kinematics Into a QP

The first-order differential kinematics for a serial-link manipulator are described by

$$\boldsymbol{\nu}(t) = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}(t) \tag{2}$$

where $\boldsymbol{\nu}(t) = \begin{pmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{pmatrix}^\top \in \mathbb{R}^6$ is the end-effector spatial velocity and $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6 \times n}$ is the manipulator Jacobian which relates $\boldsymbol{\nu}$ to the joint velocities $\dot{\boldsymbol{q}}(t)$, and n is the number of joints in the robot.

We can incorporate (2) into a quadratic program as

$$\min_{\dot{\boldsymbol{q}}} \quad f_o(\dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^\top \boldsymbol{I}\dot{\boldsymbol{q}},$$

$$\text{subject to} \quad \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{\nu},$$
$$\dot{\boldsymbol{q}}^- \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}^+ \tag{3}$$

where $\boldsymbol{I}$ is an $n \times n$ identity matrix, $\dot{\boldsymbol{q}}^-$ and $\dot{\boldsymbol{q}}^+$ represent the upper and lower joint-velocity limits, and no inequality constraints need to be defined. If the robot has more degrees-of-freedom than necessary to reach its entire task space, the QP in (3) will achieve the desired end-effector velocity with the minimum joint-velocity norm. However, there are other things we can optimise for, such as manipulability.

B. Incorporating Manipulability Into a QP

The manipulability measure, devised in [1], describes the conditioning of a manipulator, or how easily the robot can achieve any arbitrary velocity. We observe that maximising the spatial translational velocity performance of the robot is more important than rotational performance. Acknowledging this, and to avoid mixing units and associated non-homogeneity issues, we adopt a strategy similar to [18] and use only the three rows of the manipulator Jacobian that correspond to the translational velocity $\boldsymbol{J}_t(q)$. The translational manipulability measure is a scalar

$$m_t = \sqrt{\det\left(\boldsymbol{J}_t(\boldsymbol{q})\boldsymbol{J}_t(\boldsymbol{q})^\top\right)} \tag{4}$$

which we can consider as a measure of the distance to a kinematic singularity. A symptom of approaching a singularity is that the required joint velocities reach impossible levels [15] due to the poorly conditioned Jacobian. For reactive control in dynamic environments the motion required at the next step is not known ahead of time, so the ability to achieve arbitrary task-space velocity is critical as will be demonstrated in Sec. VIII-A.

We can use the time derivative of (4) and incorporate it into the QP in (3) to maximise manipulability while still achieving the desired end-effector velocity [11]

$$\min_{\dot{\boldsymbol{q}}} \quad f_o(\dot{\boldsymbol{q}}) = \underbrace{\frac{1}{2}\dot{\boldsymbol{q}}^\top (\lambda_q\boldsymbol{I})\,\dot{\boldsymbol{q}}}_{\text{A}} \underbrace{-\boldsymbol{J}_m^\top\dot{\boldsymbol{q}}}_{\text{B}},$$

$$\text{subject to} \quad \underbrace{\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{\nu}}_{\text{C}},$$

$$\underbrace{\dot{\boldsymbol{q}}^- \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}^+}_{\text{D}} \tag{5}$$

A $-$ Minimises the norm of the velocity

B $-$ Maximises the manipulability of the robot

C $-$ Ensures the desired end-effector velocity

D $-$ Ensures the the robot's velocity limits are respected

where λ_q is the gain for velocity-norm minimisation, and $\boldsymbol{J}_m \in \mathbb{R}^n$ is the translational manipulability Jacobian which relates the rate of change of manipulability to the joint velocities

$$\boldsymbol{J}_m^\top = \begin{pmatrix} m_t \, \text{vec}\left(\boldsymbol{J}_t\boldsymbol{H}_t^\top\right)^{\top\top} \text{vec}\left((\boldsymbol{J}_t\boldsymbol{J}_t^\top)^{-1}\right) \\ m_t \, \text{vec}\left(\boldsymbol{J}_t\boldsymbol{H}_{t2}^\top\right)^{\top\top} \text{vec}\left((\boldsymbol{J}_t\boldsymbol{J}_t^\top)^{-1}\right) \\ \vdots \\ m_t \, \text{vec}\left(\boldsymbol{J}_t\boldsymbol{H}_t^\top\right)^{\top\top} \text{vec}\left((\boldsymbol{J}_t\boldsymbol{J}_t^\top)^{-1}\right) \end{pmatrix} \tag{6}$$

with $\boldsymbol{J}_m^\top \in \mathbb{R}^n$ and where the vector operation $\text{vec}(\;): \mathbb{R}^{a \times b} \to \mathbb{R}^{ab}$ converts a matrix column-wise into a vector, and $\boldsymbol{H}_{ti} \in \mathbb{R}^{3 \times n}$ is the ith translational component of the manipulator Hessian tensor $\boldsymbol{H} \in \mathbb{R}^{6 \times n \times n}$ [19].

IV. MODELLING OBSTACLES

A point in 3D space can be represented as $\boldsymbol{p} \in \mathbb{R}^3$. The distance d between a point $\boldsymbol{p}_r$ on a robot and a point $\boldsymbol{p}_o$ on an obstacle is

$$d_{ro} = ||\boldsymbol{p}_o - \boldsymbol{p}_r|| \tag{7}$$

and a unit vector $\hat{\boldsymbol{n}}_{ro}$ pointing from $\boldsymbol{p}_r$ to $\boldsymbol{p}_o$ is

$$\hat{\boldsymbol{n}}_{ro} = \frac{\boldsymbol{p}_r - \boldsymbol{p}_o}{d_{ro}} = -\hat{\boldsymbol{n}}_{or} \in \mathbb{R}^3. \tag{8}$$

The time derivative of (8) is

$$\dot{d}_{ro} = \frac{\mathrm{d}}{\mathrm{d}t}||\boldsymbol{p}_o(t) - \boldsymbol{p}_r(t)||$$
$$= \hat{\boldsymbol{n}}_{or}^\top (\dot{\boldsymbol{p}}_o(t) - \dot{\boldsymbol{p}}_r(t)) \tag{9}$$

We know from differential kinematics that the end-effector velocity is $\boldsymbol{\nu}(t) = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}(t)$. Furthermore, we can calculate the

velocity of any point fixed to the robot by taking the translational-velocity component of a modified manipulator Jacobian that considers the point as the robot's end-effector

$$\boldsymbol{J}_p(\boldsymbol{q}_{0 \quad k}) = \left(\frac{\partial}{\partial \boldsymbol{q}_{0 \quad k}} \left({}^0\boldsymbol{T}_{\boldsymbol{q}_k} \bullet {}^{\boldsymbol{q}_k}\boldsymbol{T}_p \right) \right)$$

$$\boldsymbol{J}_p(\tilde{\boldsymbol{q}}) = \left(\frac{\partial}{\partial \tilde{\boldsymbol{q}}} \left({}^0\boldsymbol{T}_{\boldsymbol{q}_k} \bullet {}^{\boldsymbol{q}_k}\boldsymbol{T}_P \right) \right) \tag{10}$$

where k is the index of the link to which p_r is attached, $\boldsymbol{J}_p \in {}^{3 \times k}$ is the Jacobian relating the velocity of the point $\boldsymbol{p}$ to the velocities of joints 0 to k, ${}^0\boldsymbol{T}_{\boldsymbol{q}_k}$ describes the pose of link k with respect to the base frame of the robot, $\bullet$ represents composition, and $(\ )$ is a function that converts the partial derivative of the pose from a tensor ${}^{6 \times k \times k}$ to the translational velocity Jacobian $\in {}^{3 \times k}$. Note that this Jacobian is only a function of the k joints preceding the link to which the point is attached. We denote the set of joints which $\boldsymbol{J}_p$ depends on as $\tilde{\boldsymbol{q}}$ which is of a variable size – if the point is attached to the end-effector then $k = n$.

The pose ${}^{\boldsymbol{q}_k}\boldsymbol{T}_P$ a frame P fixed to link k, with respect to the reference frame of link k, and whose origin is $\boldsymbol{p}_r$. The orientation of this frame is arbitrary, although the direction $\hat{\boldsymbol{n}}$ is expressed in this frame, so we disregard the angular velocity component from the partial derivative in (10). More details on calculating this Jacobian can be found in [19]. Our provided software package [2] makes it trivial to calculate these Jacobians.

Using (10), we can now write

$$\dot{\boldsymbol{p}}_r(t) = \boldsymbol{J}_{p_r}(\tilde{\boldsymbol{q}})\dot{\tilde{\boldsymbol{q}}}(t) \tag{11}$$

and the rate of change of distance becomes

$$\dot{d}_{ro}(t) = \hat{\boldsymbol{n}}_{or}^\top \left(\dot{\boldsymbol{p}}_o(t) - \boldsymbol{J}_{p_r}(\tilde{\boldsymbol{q}})\dot{\tilde{\boldsymbol{q}}}(t) \right)$$
$$= \hat{\boldsymbol{n}}_{or}^\top \dot{\boldsymbol{p}}_o(t) - \hat{\boldsymbol{n}}_{or}^\top \boldsymbol{J}_{p_r}(\tilde{\boldsymbol{q}})\dot{\tilde{\boldsymbol{q}}}(t). \tag{12}$$

The coefficient of $\dot{\tilde{\boldsymbol{q}}}(t)$ is the distance Jacobian

$$\boldsymbol{J}_d(\tilde{\boldsymbol{q}}) = \hat{\boldsymbol{n}}_{ro}^\top \boldsymbol{J}_{p_r}(\tilde{\boldsymbol{q}}) \in {}^{6}. \tag{13}$$

and substituting (13) into (12) we obtain the scalar

$$\boldsymbol{J}_d(\tilde{\boldsymbol{q}})\dot{\tilde{\boldsymbol{q}}}(t) = \dot{d}_{ro}(t) - \hat{\boldsymbol{n}}_{or}^\top \dot{\boldsymbol{p}}_o(t) \tag{14}$$

which is now in a form usable for our QP.

V. OBSTACLE AVOIDANCE

A. Velocity Dampers

The velocity damper approach outlined in [12] prevents robot failure by damping or restricting a velocity before limits are reached. This can be used to restrict joint motion before hitting a limit, or to constrain robot velocity before hitting an obstacle. The general velocity damper formula is

$$v \leq \xi \frac{d - d_s}{d_i - d_s} \tag{15}$$

where v is the rate of change of the distance d, ξ is a positive gain which adjusts the aggressiveness of the damper, d_i is the influence distance during which the damper is active, and d_s is the stopping distance which d will never be less than. These distances are illustrated in Fig. 2. When used within an optimiser,
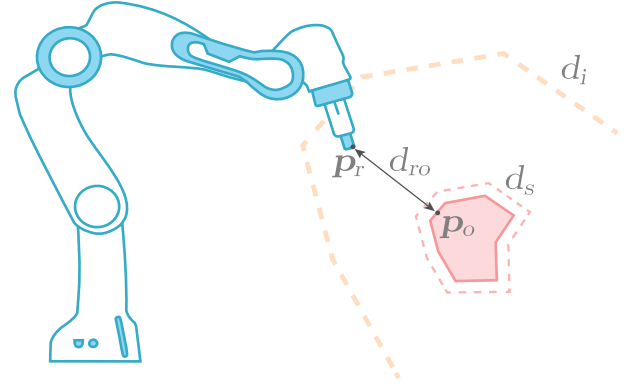

Fig. 2. The distance ${}_{ro}$ from a point on the robot ${}_r$ to a point on an obstacle ${}_o$ starts to affect the robot motion once ${}_{ro}$ is less than the influence distance . The velocity damper ensures that ${}_{ro}$ is never less than the stopping distance ${}_s$.

v must be found which is less than the limit set by the velocity damper.

We can incorporate obstacle avoidance into the velocity damper as

$$\dot{d}_{ro}(t) \leq \xi \frac{d - d_s}{d_i - d_s}. \tag{16}$$

However, to use this as an inequality constraint of a QP we must incorporate the form of (14)

$$\boldsymbol{J}_d(\tilde{\boldsymbol{q}})\dot{\tilde{\boldsymbol{q}}}(t) \leq \xi \frac{d - d_s}{d_i - d_s} - \hat{\boldsymbol{n}}_{or}^\top \dot{\boldsymbol{p}}_o(t). \tag{17}$$

The effect of (17) is that the optimiser must choose joint velocities which cause the rate of change of the minimum distance between the robot and the obstacle to increase. Consequently, the point on the robot and the point on the obstacle will never collide while the damper is active. By adding multiple inequality constraints in the form of (17), the robot can respond to infinitely many dynamic obstacles simultaneously. Multiple obstacles can be incorporated into the same inequality constraint by stacking instances of (17) vertically as

$$\boldsymbol{A}\dot{\boldsymbol{q}}(t) \leq \boldsymbol{b}$$

$$\begin{pmatrix} \boldsymbol{J}_d\left(\tilde{\boldsymbol{q}}_0\right) & {}_{1 \times n-k} \\ \vdots & \vdots \\ \boldsymbol{J}_{d_l}(\tilde{\boldsymbol{q}}_l) & {}_{1 \times n-k_l} \end{pmatrix} \dot{\boldsymbol{q}}(t) \leq \begin{pmatrix} \xi_0 \frac{d - d_s}{d - d_s} - \hat{\boldsymbol{n}}_{or}^\top \dot{\boldsymbol{p}}_o(t) \\ \vdots \\ \xi_l \frac{d_l - d_s}{d - d_s} - \hat{\boldsymbol{n}}_{or_l}^\top \dot{\boldsymbol{p}}_{o_l}(t) \end{pmatrix} \tag{18}$$

for l obstacles, where the Jacobians within $\boldsymbol{A} \in {}^{l \times n}$ have been stacked and padded with zeros to make them of constant length n. Obviously, given multiple obstacles and finite robot degrees-of-freedom, there will exist situations for which there is no viable solution.

If the equality constraint in (5) is active, the robot will be unable to diverge from its path to dodge an obstacle and the optimiser will fail – it is unable to satisfy both the equality and inequality constraints. To circumvent this, we will add *slack* to our optimiser.

B. Adding Slack

We can augment our optimisation variable $\dot{\boldsymbol{q}}$ by adding a slack vector $\in \mathbb{R}^6$ and change the problem to

$$\boldsymbol{\nu}(t) - (t) = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}(t) \tag{19}$$

where represents the difference between the desired and actual end-effector velocity – relaxing the trajectory constraint. We reformulate our QP problem as

$$\min_x \quad f_o(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \mathcal{Q}\boldsymbol{x} + \mathcal{C}^\top \boldsymbol{x},$$

$$\text{subject to} \quad \mathcal{J}\boldsymbol{x} = \boldsymbol{\nu},$$

$$\mathcal{A}\boldsymbol{x} \leq \mathcal{B}$$

$$\mathcal{X}^- \leq \boldsymbol{x} \leq \mathcal{X}^+ \tag{20}$$

where

$$\boldsymbol{x} = \begin{pmatrix} \dot{\boldsymbol{q}} \\ \end{pmatrix} \in \mathbb{R}^{n+6)} \tag{21}$$

$$\mathcal{Q} = \begin{pmatrix} _q\boldsymbol{I}_{n\times n} & _{6\times 6} \\ _{n\times n} & \boldsymbol{I}_{6\times 6} \end{pmatrix} \in \mathbb{R}^{n+6)\times n+6)}, \tag{22}$$

$$\mathcal{J} = \begin{pmatrix} \boldsymbol{J} & \boldsymbol{I}_{6\times 6} \end{pmatrix} \in \mathbb{R}^{6\times n+6)} \tag{23}$$

$$\mathcal{C} = \begin{pmatrix} \boldsymbol{J}_m \\ _{6\times 1} \end{pmatrix} \in \mathbb{R}^{n+6)} \tag{24}$$

$$\mathcal{A} = \begin{pmatrix} \boldsymbol{J}_d\,(\tilde{\boldsymbol{q}}_0) & _{1\times 6+n-k} \\ \vdots & \vdots \\ \boldsymbol{J}_{d_l}(\tilde{\boldsymbol{q}}_l) & _{1\times 6+n-k_l} \end{pmatrix} \in \mathbb{R}^{l\times n+6)} \tag{25}$$

$$\mathcal{B} = \begin{pmatrix} \xi\frac{d-d_s}{d-d_s} - \hat{\boldsymbol{n}}_{or}^\top\,\dot{\boldsymbol{p}}_o\,(t) \\ \vdots \\ \xi\frac{d_l-d_s}{d-d_s} - \hat{\boldsymbol{n}}_{or_l}^\top\,\dot{\boldsymbol{p}}_{o_l}(t) \end{pmatrix} \in \mathbb{R}^l \tag{26}$$

$$\mathcal{X}^{-,+} = \begin{pmatrix} \dot{\boldsymbol{q}}^{-,+} \\ ^{-,+} \end{pmatrix} \in \mathbb{R}^{n+6}. \tag{27}$$

We can augment the matrices $\mathcal{A}$ and $\mathcal{B}$ with velocity dampers to ensure the controller respects joint position limits. We replace $\mathcal{A}$ and $\mathcal{B}$ with

$$\mathcal{A}^* = \begin{pmatrix} \mathcal{A} \\ \boldsymbol{I}_{n\times n+6} \end{pmatrix} \in \mathbb{R}^{l+n\times n+6)} \tag{28}$$

$$\mathcal{B}^* = \begin{pmatrix} \mathcal{B} \\ \eta\frac{\rho-\rho_s}{\rho-\rho_s} \\ \vdots \\ \eta\frac{\rho_n-\rho_s}{\rho-\rho_s} \end{pmatrix} \in \mathbb{R}^{l+n} \tag{29}$$

where ρ represents the distance to the nearest joint limit (could be an angle for a revolute joint), ρ_i represents the influence distance in which to operate the damper, and ρ_s represents the minimum distance for a joint to its limit. It should be noted that each row

β	Adjusts how fast the robot's end-effector will travel towards the goal pose. A small value will increase the time taken to reach the goal while a large value will require large joint velocities and consequently limit the flexibility of the optimiser to achieve sub-goals.
λ_q	Adjusts the trade-off between minimising the joint velocity norm compared to maximising manipulability. A large value requires large joint velocities while a small value reduces the possible manipulability gains.
λ_δ	Adjusts the cost of increasing the norm of the slack vector. A gain too large limits the possible additional manipulability achievable, while a gain too small leads to the possibility that the slack will cancel out the desired velocity, leaving a steady-state error.
ξ	Adjusts how aggressively the robot will be repelled from obstacles. Too large results in large joint velocities, while too small will reduce obstacle avoidance performance.
η	Adjusts how aggressively the robot will be repelled from joint limits. Too large results in large joint velocities, while too small will reduce joint-limit-avoidance performance.

in $\mathcal{A}^*$ and $\mathcal{B}^*$ is only added if the distance, d and ρ, is less than the corresponding influence distance, d_i and ρ_i. When a row is added to those matrices, the velocity damper for the collision avoidance or joint limit avoidance is activated.

VI. PROPOSED CONTROLLER

Our proposed controller, NEO, exploits the QP described by (20) within a position-based servoing (PBS) scheme. This controller seeks to drive the robot's end-effector in a straight line from its current pose to a desired pose. PBS is formulated as

$$\boldsymbol{\nu}_e = \quad \psi\left(({}^0\boldsymbol{T}_e)^{-1} \bullet {}^0\boldsymbol{T}_{e^*}\right) \tag{30}$$

where $\in \mathbb{R}^+$ is a gain term, ${}^0\boldsymbol{T}_e \in \mathbf{SE}(3)$ is the current end-effector pose in the robot's base frame, ${}^0\boldsymbol{T}_{e^*} \in \mathbf{SE}(3)$ is the desired end-effector pose in the robot's base frame, and $\psi()\colon \mathbb{R}^{4\times 4} \mapsto \mathbb{R}^6$ is a function which converts a homogeneous transformation matrix to a spatial twist.

By setting $\boldsymbol{\nu}$ in (20) equal to $\boldsymbol{\nu}_e$ in (30) the robot will be driven towards the goal, however, not necessarily in a straight line. The robot will diverge from the path to improve manipulability, avoid joint velocity limits, avoid joint position limits, and dodge both static and dynamic obstacles. We detail the effects of changing the controller parameters in Table I.

The proposed controller is subject to local minima which can prevent it from reaching the goal pose if the direct path is blocked by an obstacle. We coded a simple *retreat* heuristic that has proven effective for many common cases: if the direct path between the end-effector and the goal is blocked by obstacles and the closest obstacle is within the influence distance d_i, then we bias v_x and v_y components of the desired end-effector velocity $\boldsymbol{\nu}_e$ to make the arm retract towards the robot's base frame.

We use our Python library, the Robotics Toolbox for Python [2], to implement our controller. We use the Python library qpsolvers which implements the quadratic programming solver [21] to optimise (20).

VII. Experiments

We validate and evaluate our controller by testing on a real manipulator as well as in simulation. In these experiments, NEO has full state information of the environment which otherwise would need to be provided by eye in hand and/or a third-person-view camera with depth sensing capabilities.

We use the following values for the controller parameters: $\xi = 1$, $d_i = 0.3\,\text{m}$, and $d_s = 0.05\,\text{m}$ in (26), $\eta = 1$, $\rho_i = 50°$, and $\rho_s = 2°$ in (29), $= 1$ in (30), and $_q = 0.01$, and $= \frac{1}{e}$ in (22), where e represents the total error between the current and desired end-effector pose. By setting to be inversely proportional to pose error, the optimiser has freedom to maximise manipulability at the beginning of the trajectory while the increasing restriction ensures the end-effector continues to approach the goal. Table I describes each of the gains within NEO, as well as the consequence of making them too large or too small. In practice we found that performance was generally sound within 50% to 200% of the chosen gain values. As discussed in Section VI, a velocity damper is incorporated into the controller when the minimum distance between the robot collision model and the obstacle is less than the influence distance d_i. This means there may be several velocity dampers for a single collision object, each with respect to a different robot link collision model.

For the simulated experiments, we use our open-source simulator Swift [20] with our Python library [2] to simulate the Willow Garage PR2. For the physical experiments, we use our Python library [2], ROS middleware to interface with the robot, and the 7 °-of-freedom Franka-Emika Panda robot.

A. Experiment 1: Physical Robot

We show the merits of NEO by having it operate on a physical Panda manipulator for several different scenarios. The tasks reflect grasping scenarios where the end-effector starts high above a table, and the desired end-effector pose is just above the table. However, the robot must dodge one or more non-stationary obstacles present in the environment. This reflects a scenario where a robot is operating in close proximity to humans or other robots. We compare the performance of NEO in these tasks to the reactive potential field approach (PF) [13], [14]. The scenarios tested are as follows

    a) The robot must servo to a hypothetical grasp target on the table while a sphere of 0.05 m radius travels towards the robot with a velocity of 0.2 m/s. If no action is taken, the sphere will collide with the robot's end-effector.

    b) The same as above, except there is a second sphere of 0.05 m radius travelling towards the robot's elbow with a velocity of 0.2 m/s. If no action is taken, the first sphere will collide with the robots end-effector, while the second collides with the elbow.

    c) The same as above, except the desired end-effector pose changes by translating across the table with a velocity of 0.1 m/s for 4 seconds during the task. This represents a scenario where the grasp target has been disturbed. If no action is taken, the robot will collide with both spheres.

B. Experiment 2: Motion Planning Benchmark

We compare NEO to several state-of-the-art motion planning algorithms on 198 problems over five different test scenes displayed in Fig. 3. The test scenes were part of a selection defined in the MoveIt benchmark [22], [23]. However, the five we selected: Bookshelves, Counter Top, Industrial A, Industrial B, and Tunnel, were the most complex.

In each problem, the robot is initialised in a defined joint configuration. The final position of the robot is defined by another joint configuration. The motion planners must plan between the initial and final configurations. However, NEO does not work by constraining the final joint configuration as it goes against the premise of task-space reactive control to know the final joint configuration. Therefore, NEO is constrained by the end-effector pose of the robot from the forward kinematics of the specified final joint configuration.

We obtained the definition of the benchmark scenes, initial and target configurations, and motion planning implementations from Trajopt's supplementary material [5]. We compare our results to four motion planners: Trajopt [5], OMPL-RRTConnect [24], and OMPL-LBKPIECE [25], and CHOMP [4]. As expected, the potential field approach performed extremely poorly in this experiment due to local minima, and those results have been excluded.

VIII. Results

The average execution time of the NEO controller during the experiments was 9.8 ms using an Intel i7-8700 K CPU with 12 cores at 3.70 GHz in a single threaded process – a control rate just above 100 Hz. Each collision model to obstacle pair being considered in the controller (i.e. the minimum distance between them is less than d_i) takes 0.5 ms. While this is an order-of-magnitude slower than vanilla RRMC it provides significant additional capability at a rate well in excess of typical vision sensors which have a refresh rate of 30 Hz – our controller is much faster than the slowest part of the pipeline. Reducing the execution time is possible since many parts of the controller, such as collision checking, are parallelizable.

A. Experiment 1 Results

Scenario a) displayed in Fig. 4(a), shows that for both controllers, once the obstacle gets close to the robot, the robot's end-effector stalls in its progress towards the goal. The PF controller initially repels the robot from the obstacle but eventually collides as the robot was not able to move around it. The NEO controller works correctly and dodges the obstacle while never coming closer than the stopping distance of 5 cm.

Scenario b) displayed in Fig. 4(b) is more difficult, as there are two obstacles on a collision course with the robot. The PF controller once again starts to dodge the obstacles but collides after 6 seconds. The NEO controller, while being disrupted twice from the desired trajectory, dodges both obstacles and achieves the goal pose.

Scenario c) displayed in Fig. 4(c), shows similar results to scenario b), where the PF controller is initially repelled from
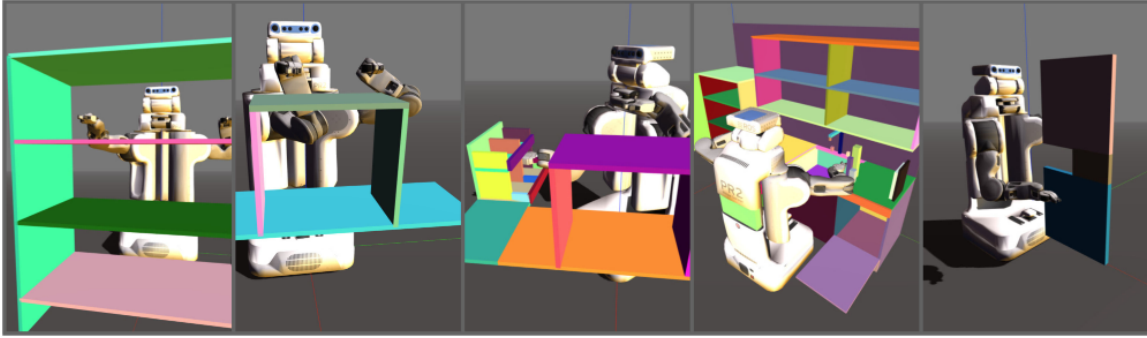
Fig. 3. The five different test scenes in the motion planning benchmark: Bookshelf, Industrial A, Industrial B, Counter, and Tunnel. Images are created using Swift [20] and the Robotics Toolbox for Python [2].

(a) Experiment 1a

(b) Experiment 1b

(c) Experiment 1c
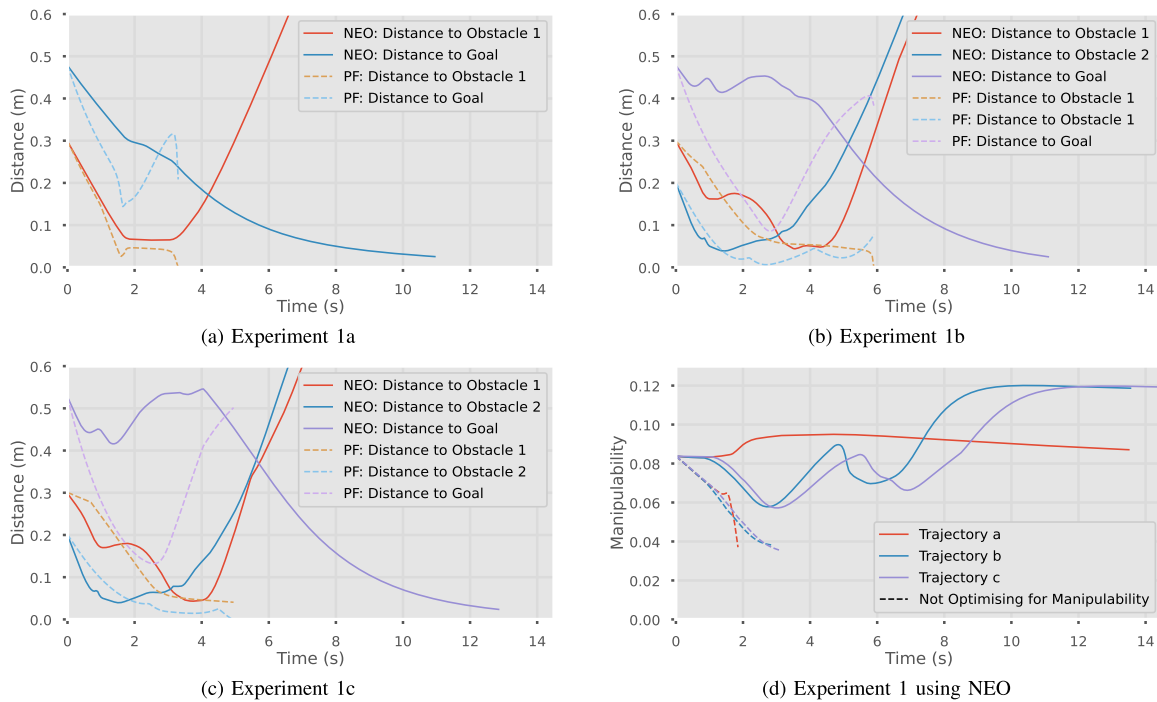
(d) Experiment 1 using NEO

Fig. 4. (a)–(c) Euclidean distance between robot and obstacles, and robot and goal comparing NEO to the potential field (PF) method. (d) The manipulability of the robot being controlled using NEO, throughout the 3 scenarios.

the obstacles before colliding with them and the NEO controller achieves the goal.

Fig. 4(d) displays the manipulability of the robot during each trajectory with and without manipulability optimisation. With manipulability optimisation, despite the complex nature of the robot movement to dodge dynamic obstacles and track a moving goal pose, the manipulability of the robot remains very high in all three trajectories. Without manipulability optimisation, the manipulability of the robot drops rapidly in all three scenarios. Consequently, when the robot needs large spatial velocities to dodge the dynamic obstacles, it is unable to stay within the joint velocity limits and this leads to failure. This clearly demonstrates that manipulability is an important factor for robust reactive robot performance.

In general, controller failure is caused by the optimiser failing to find a solution that respects the joint velocity limits

of the robot. Large and/or fast-moving obstacles increase the likelihood of this happening due to the large spatial velocities required to avoid collisions. This experiment shows that a low value of robot manipulability is strongly correlated with this mode of optimiser failure.

B. Experiment 2 Results

The results from Experiment 2 are displayed in Table II and show that NEO performance has a competitive success rate compared to some well-known planners. NEO scored very well in the *Bookshelf* scene, successfully completing 45 out of 45 tasks. Additionally, in the *Industrial A* scene, NEO completed 61 out of 66 tasks (92% success rate), but performed less well in the other scenes. From Fig. 3, it is clear that the *Bookshelf* and *Industrial A* scenes contain many large broad obstacles, while

TABLE II
EXPERIMENT 2: MOTION PLANNING BENCHMARK

	NEO (ours)	Trajopt	OMPL-RRTConnect	OMPL-LBKPIECE	CHOMP-HMC
Success	74.2%	81.8%	85.4%	75.8%	65.2%
Average time (ms)	9.8	191	615	1300	4910

Industrial B and *Counter* scenes contain many small obstacles with numerous small and enclosed spaces. NEO performs less well on this latter category.

Our goal in this work has been reactivity not complex motion planning per se. NEO does not perform as well as the best of the planners but for scenes which do not feature many intricate obstacles with small confined spaces it is clearly a viable alternative, and with the added benefits of reactivity such as dynamic obstacle and joint limit avoidance as well as manipulator conditioning. NEO would ideally be employed as a local controller in conjunction with a global planner that provides the global trajectory.

Another key result from Table II is that NEO takes around 10 ms to compute the next action, while the motion planners must compute the whole trajectory before the robot can move. This ranges from 191 ms to 4910 ms for the motion planners. Consequently, they lack the reactivity required to be able to respond to dynamic obstacles, changes in the environment, or even changes in the goal pose. When testing NEO on the benchmark without optimising for manipulability, we obtained the same score of 74.2%. This is not surprising since, unlike in Experiment 1, the obstacles are static and their influence on the control problem grows slowly as the robot moves.

IX. CONCLUSION

In this letter we have presented NEO, a novel Expeditious Optimisation Algorithm for manipulator motion control while dodging dynamic and static obstacles, avoiding joint limits, and maximising manipulability. Our experiments show the purely reactive and real-time NEO controller is reliable in environments containing dynamic obstacles reflective of workspaces containing multiple robots working with humans. Additionally, NEO is shown to be viable for general environments containing benches or shelves but will struggle in environments containing small confined spaces with confined spaces and is an area for future work. In such scenarios, NEO is better employed as a local controller when dynamic obstacles are encountered. NEO has a strong advantage in terms of low upfront computation time.

Our approach requires the computation of the manipulability Jacobian, the manipulator Jacobian (between any points on the robot), the manipulator Hessian, and distance Jacobians. The associated Python library, the Robotics Toolbox for Python [2], can conveniently compute all these values. The only required input for our library is a kinematic robot model which can be in standard or modified Denavit-Hartenberg notation, a URDF/XACRO model, or the ETS format [19].

REFERENCES

[1] T. Yoshikawa, "Manipulability of robotic mechanisms," *Int. J. Robot. Res.*, vol. 4, no. 2, pp. 3–9, 1985.

[2] J. Haviland and P. Corke, "Robotics toolbox python software package." [Online]. Available: https://github.com/petercorke/robotics-toolbox-python

[3] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 4569–4574.

[4] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 489–494.

[5] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Robot.: Sci. Syst.*, vol. 9, no. 1, 2013, pp. 1–10.

[6] S. K. Chan and P. D. Lawrence, "General inverse kinematics with the error damped pseudoinverse," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1988, pp. 834–839.

[7] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Trans. Man-Mach. Syst.*, vol. 10, no. 2, pp. 47–53, Jun. 1969.

[8] J. Park, W. Chung, and Y. Youm, "Computation of gradient of manipulability for kinematically redundant manipulators including dual manipulators system," *Trans. Control, Automat. Syst. Eng.*, vol. 1, no. 1, pp. 8–15, 1999.

[9] S. Patel and T. Sobh, "Manipulator performance measures-a comprehensive literature survey," *J. Intell. Robotic Syst.*, vol. 77, no. 3-4, pp. 547–570, 2015.

[10] D. J. Agravante, G. Claudio, F. Spindler, and F. Chaumette, "Visual servoing in an optimization framework for the whole-body control of humanoid robots," *IEEE Robot. Automat. Lett.*, vol. 2, no. 2, pp. 608–615, Apr. 2017.

[11] J. Haviland and P. Corke, "A purely-reactive manipulability-maximising motion controller," 2020, *arXiv:2002.11901*.

[12] B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom," in *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 4, 1987, pp. 1152–1159.

[13] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Auton. Robot Veh*. Berlin, Germany: Springer, 1986, pp. 396–404.

[14] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *Proc. IEEE Int. Conf. Humanoid Robots*, 2008, pp. 91–98.

[15] P. Corke, Robotics, *Vision and Control*, 2nd ed. Springer International Publishing, 2017.

[16] D. Morrison, J. Leitner, and P. Corke, "Closing the loop for robotic grasping: A. real-time, generative grasp synthesis approach," in *Robotics: Science and Systems XIV*, Jun. 2018.

[17] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Science & Business Media, 2006.

[18] P. Cardou, S. Bouchard, and C. Gosselin, "Kinematic-sensitivity indices for dimensionally nonhomogeneous jacobian matrices," *IEEE Trans. Robot.*, vol. 26, no. 1, pp. 166–173, Feb. 2010.

[19] J. Haviland and P. Corke, "A systematic approach to computing the manipulator differential kinematics using the elementary transform sequence," 2021, *arXiv:2010.08696*.

[20] J. Haviland and P. Corke, "Swift: A python/javascript robotics simulator." [Online]. Available: https://github.com/jhavl/swift

[21] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Math. Program.*, vol. 27, no. 1, pp. 1–33, 1983.

[22] S. Chitta, I. Sucan, and S. Cousins, "Moveit [ros topics]," *IEEE Robot. Automat. Mag.*, vol. 19, no. 1, pp. 18–19, Mar. 2012.

[23] B. Cohen, I. A. Şucan, and S. Chitta, "A generic infrastructure for benchmarking motion planners," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 589–595.

[24] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. Int. Conf. Robot. Automat.*, vol. 2, 2000, pp. 995–1001.

[25] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Found. Robot. VIII*. Berlin, Germany: Springer, 2009, pp. 449–464.