QCQP-TUNNELING: ELLIPSOIDAL CONSTRAINED AGENT NAVIGATION

Sanjeev Sharma
Indian Institute of Technology Roorkee,
Roorkee, 247667, India
email: sanjeev@searching-eye.com *

ABSTRACT
This paper presents a convex-QCQP based novel path planning algorithm named ellipsoidal constrained agent navigation (ECAN), for a challenging problem of online path planning in completely unknown and unseen continuous environments. ECAN plans path for the agent by making a tunnel of overlapping ellipsoids, in an online fashion, through the environment. Convex constraints in the ellipsoid-formation step circumvent collision with the obstacles. The problem of online-tunneling is solved as a convex-QCQP. This paper assumes no constraints on shape of the agent and the obstacles. However, to make the approach clearer, this paper first introduces the framework for a point-mass agent with point-size obstacles. After explaining the underlying principle in drawing an ellipsoid tunnel, the framework is extended to the agent and obstacles having finite area (2d space) and finite-volume (3d-space).

KEY WORDS
Path Planning, Mobile Robot/Vehicle Navigation, Convex Optimization, Continuous Environment.

1 Introduction

Path planning is a vital component of autonomous navigation. Despite the recent success of DARPA Urban Challenge, path planning in completely unknown, unseen and continuous spaces is still a challenging problem. This paper presents a convex-QCQP ([1]) formulation based novel algorithm (ECAN) for path planning in completely unknown, unseen and continuous 2D and 3D spaces. ECAN generates a path by creating a tunnel of overlapping ellipsoids through the environment. Convex constraints in the ellipsoid-formation step ensure obstacle avoidance. ECAN uses a point-cloud representation of surrounding obstacles to plan a path, thereby facilitating direct implementation in real-world as most of the sensors (image, 3D-Lidar etc.) return point cloud representations of surrounding obstacles.

When the environment is known or partially known, RRT & spline based planners ([2],[3]) and grid-based planners & re-planners ([4],[5],[6],[7]) can be used. Recent research has focused on extending grid-planners to continuous heading directions ([6],[7],[8]) since discretization often results in suboptimal path even when the environment is known. A reasonable approach to plan in continuous spaces is to use optimization techniques. In recent years, with increasing computational resources, a lot of progress has been made in path planning using the MILP, both in known and partially known environments ([9],[10],[11],[12]). MILP is a powerful mathematical programming tool that can handle collision avoidance constraints and provide an efficient path through the environment. However, computational requirements with MILP based planners grow too high even in known environments with reasonable number of constraints ([11],[12]), making them practically less significant. This paper presents an unknown and unseen spaces path planning algorithm, that scales well with number of obstacles, using convex-optimization ([1]). Convex optimization has been widely accepted as a powerful tool to solve many engineering problems. Convex functions, most importantly, do not have the pitfall of local minima, assuring that the achieved solution always happens to be the optimal. Despite this, direct implementation of convex optimization for path planning, like MILP, is a less explored area. To the best of our knowledge, this is the first ever work using convex optimization (in this case a QCQP) as a main planner for path planning in completely unknown and unseen spaces. ECAN utilizes only the information in the field-of-view (fov) of the agent to plan (online) a path leading to the goal location. Also, unlike most of the literature in path planning, the proposed algorithm (ECAN) does not restrict the agent/vehicle to a point-mass entity and can directly handle the agent having finite-area/volume with any shape. Simulation results show feasibility of ECAN in UGV and UAV (helicopters/fixed-wing planes) path planning. The paper proceeds as follows: sec-2 presents related work; sec-3 outlines the underlying principle of ECAN with point-agent and point-obstacles; sec-4 outlines notations, describes ECAN framework and navigation strategy with point-masses; sec-5 explains finite agent & obstacle architecture; and sec-6 and 7 present experiments and conclusion & future work respectively.

2 Related Work

A 2D Tunnel-MILP algorithm is presented in [12] for planning a path through a sequence of pre-decided convex polytopes. However, this method is computationally intensive, and it works only in known & seen environments. A sim-

*Author earned his B.Tech in Electrical Engineering from Indian Institute of Technology Roorkee, 2011, and did this research as an undergraduate student.

ilar work by Blackmore [13] proposes UAV path planning with stochastic uncertainty (wind) using a two-stage optimization algorithm. At the first stage, an MILP generates a feed forward map (reference control) which is then used to design a feedback (path planning with uncertainty) control, at second stage, for a linear dynamical UAV model using *chance constrained* formulation (solving SOCP, convex program). The algorithm models uncertain risks, which the present paper doesn't model, but to solve the first-stage optimization problem a deterministic and completely known environment model is necessary. Moreover, as far as UAV navigation is concerned, the paper used a constant altitude model. Vandapel *et.al.* [14] construct a 3D-network of tunnels formed by overlapping spherical bubbles through the cluttered environment. Their approach is again limited to known and seen environment and is basically a graph based approach using A* ([15]) search to decide an optimal path to the goal location through the network of tunnels. An SVM (convex program) based path planning approach is presented in [16]. The method uses the environment model, and then creates dummy obstacles around the nominal line (plane, in 3D-space) to guide the SVM to generate a desired path. A control-point insertion based spline fitting algorithm is presented in [17], taking into account the width and height of the robot. Path planning is done for ground navigation in an environment with 3D-objects. The algorithm also considers planning in non-planner surfaces by planning over mesh. An extension of this algorithm to UAV path planning would be a good approach. However, in the present version for ground navigation, the algorithm needs to know the locations of all the obstacles to find control points. In [18], a grid-based path planner plans in completely unknown spaces. The algorithm is based on straight line formalisms, and obstacles are avoided on-the-fly using a table of rules which rotates the line to avoid obstacles. Once a path is planned and the sub-goals are identified, advancing learning and trial-and-error learning improve the already planned path using the (now partially known) environment model and sub-goals. Limitations of this approach are: (i) it suffers form limited heading constraints, unlike ECAN which plans in continuous spaces; and (ii) the method can provide a resonable path only when the environment becomes (at least) partially known, while ECAN directly provides a reasonable (if not optimal, since the space is unknown and useen) path to the goal location. [19] presents an iterative MILP formulation for UAV (again constant altitude) path planning in partially known environment (with discovered obstacles known to be static) and performs tests on real flights to complete a coordinated mission. However the algorithm suffers from the computational requirements of MILP ([11],[12]). Also, to ensure safe navigation, UAV was provided with a backup trajectory and the online planner avoids obstacles following this trajectory. [20] presents a 3D-MILP planner, but requires a pre-specified path to the goal location, and is therefore limited to seen spaces.

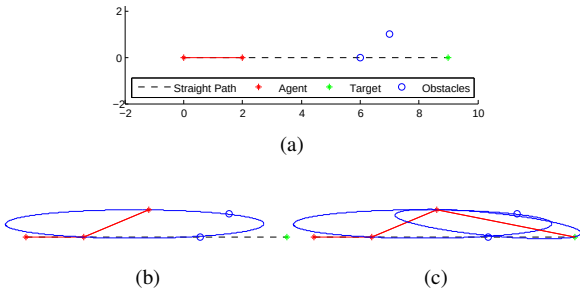


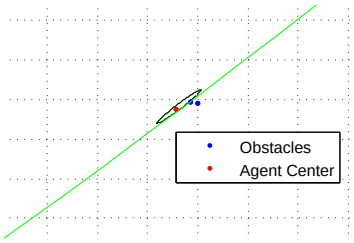
Figure 1: Basic working principle of ECAN.

3 ECAN: Underlying Planning Principle

Before beginning to describe the convex formulation in ECAN algorithm, a basic underlying principle is explained. This makes the understanding of later sections easier. For simplicity, this section considers 2D-navigation, a point-agent and static point-obstacle environment. The principle is also valid for 3D-planning. Assume an agent at $(0, 0)$ trying to reach at $(9, 0)$, see fig-1a. After navigating to $(2, 0)$ agent discovers an obstacle at $(6, 0)$. At this step an ellipsoid is formed, taking into account, the agent's location, the obstacle's location and the target location (fig-1b). Since the goal location is not lying on the boundary of the ellipsoid (indicating the presence of an obstacle), agent finds a navigation direction inside the ellipsoid (to avoid obstacle) and navigates to the safe location on the boundary of the ellipsoid (fig-1b). After navigating to the boundary, agent discovers another obstacle at $(7, 1)$. Again an ellipsoid is formed, and since the target location is on the boundary, agent navigates to this location, reaching the destination (fig-1c). Later sections describe the process of ellipsoid formation (sec-4.1), finding the navigation direction (z_n) inside the ellipsoid and the step-length (l_n) to move inside the ellipsoid (sec-4.2 and 5). Also the environment may be dynamic and moreover, in case of finite-area/volume obstacles, the agent may discover only a part of the obstacle in its field-of-view instead of the full obstacle. Therefore, due to the associated uncertainty, navigating to a safe location on the boundary of the ellipsoid may not be possible. This is tackled by finding a safe step-length to move inside the ellipsoid (sec-5.1).

4 Ellipsoidal Constrained Agent Navigation

This section presents ECAN algorithm and the underlying convex programming framework for path-planning of a point-agent navigating in an environment polulated with randomly scattered point-obstacles. The work utilizes the SDP framework for separating two data-sets by a quadratic surface [1], in this case an ellipsoid. This paper assumes that the agent has a finite field-of-view (fov) and it cannot see beyod its fov. Agent's fov is defined by parameters (r, θ) in 2D-space and by (r, θ, ϕ) in 3D space (using polar



(a)

Agent's Location = (32.83,33.78)

α	z_n	l_n^f	$\log(\lvert P^{-1} \rvert)$
0	$\begin{pmatrix}0.5891\\0.8081\end{pmatrix}$	0.1	-0.2673
0.1	$\begin{pmatrix}0.7269\\0.6868\end{pmatrix}$	.2425	-4.3858

(b)

Figure 2: (a) resulting ellipsoids with f_2 (black), and w/o f_2 (green) in convex problem-(1); (b) different parameters due to resulting ellipsoid. Ellipsoid without convex objective f_2 is too large to be depicted completely.

coordinate system), where $r \in (0, R_{fov}]$, $\theta \in [-\theta_{fov}, \theta_{fov}]$ and $\phi \in [-\phi_{fov}, \phi_{fov}]$. Parameters $(R_{fov}, \theta_{fov}, \phi_{fov})$ restrict the agent's fov. Obstacles discovered at time step t are the obstacles lying in the agent's fov. Also, agent has a local coordinate system represented as ζ_a^2 in 2D navigation and ζ_a^3 in 3D-navigation. Further details regarding the fov and implementation are in sec-5.2, this section discusses the working principle of ECAN algorithm. Throughout the paper, $z_a^t \in R^2$ (R^3, for 3D-space) represents the agent's location at time t, $z_{o_i}^t \in R^2$ (R^3, for 3D-space), $i \in \{1, ..., k\}$ represents location of the i^{th} point-obstacle at time step t and $z_g^t \in R^2$ (R^3, for 3D-space) denotes the goal location that the agent is aiming for at time t. An ellipsoid at time t is represented as $\Psi^t(P, q, r) = \{x | x^T P^t x + x^T q^t + r^t \leq 0\}$, with its interior $\textbf{int}(\Psi^t) = \{x | \Psi^t(P, q, r) < 0\}$ and $\Psi^t(z) = z^T P^t z + z^T q^t + r^t$ represents value of point z w.r.t. Ψ^t. A set of all $n \times n$ positive definite symmetric matrices is denoted as S_{++}^n. For a 2D-planning problem, $x \in R^2; P \in S_{++}^2, q \in R^2, r \in R$ and for 3D, $x \in R^3, P \in S_{++}^3, q \in R^3, r \in R$, v^T represents transpose of vector v, $|P|$ denotes determinant of a matrix $P \in S_{++}^n$, and $|q|$ denotes absolute value of $q \in R$. The minimum and maximum Eigenvalues of P are denoted as $\lambda_{min}(P)$ and $\lambda_{max}(P)$ respectively. Sec-4.1 explains QCQP formulation using the point agent and the point obstacles; sec-4.2 explains navigation inside the ellipsoid for a point-agent; and sec-5 extends the approach to the finite-area/volume agent navigating in spaces populated with the finite-area/volume obstacles.

4.1 ECAN Path Planning: Convex-QCQP Tunneling

ECAN is a 3-step online algorithm for planning in continuous spaces. The 3-steps are: (i) determining a feasible ellipsoid at time t; (ii) finding a navigation direction for navigating inside the ellipsoid; and (iii) determining the step-length to move in that direction. Each of the 3-steps requires solving a convex optimization problem. This section explains the first-step of ECAN, i.e. finding a feasible ellipsoid. An ellipsoid forming at time t (Ψ^t), parameterized by (P^t, q^t, r^t), should satisfy 3 constraints: (i) agent lies inside the ellipsoid ($\Psi(z_a^t) < 0$); (ii) obstacles lie outside the ellipsoid ($\Psi(z_{o_i}^t) > 0, i = \{1, ..., k\}$); and (iii) goal location can either be outside or on the boundary of ellipsoid ($\Psi(z_g^t) \geq 0$). These three constraints ensure a navigable space (if it exists) for the agent even in a highly cluttered environment and a constraint $P \in S_{++}^n$ results in an ellipsoid surface. Next, convex objective functions are designed such that the resulting Convex-QCQP motivates the agent to move towards the goal location.

To lure the agent to move towards goal location, the ellipsoid boundary should be as close as possible to the goal location. This is equivalent to the principal axis of resulting ellipsoid being aligned with the goal location (in ideal case with no obstacle, goal location lying in a ray from center of the ellipsoid in the direction of principal axis). This is done by adding a convex penalty term (f_1) in the objective which is $zero$ iff the goal location lies on the boundary of the resulting ellipsoid. A penalty function f_2 is added which minimizes distance of current location of the agent from the boundary of resulting ellipsoid. At first this may seem counter-intuitive, but eventually it leads to a stable ellipsoid. Without f_2, Ψ^t can grow unboundedly large in any direction to satisfy the convex constraints. This can result in a very small navigable distance in the feasible navigation direction inside the ellipsoid. Fig-2a shows two ellipsoids, one determined using f_2 (black) and other (green) without f_2 (see, table in fig-2b and sec-4.2 for further details). A third penalty function f_3 is added to make a locally maximal ellipsoid around the agent, taking into account the surrounding obstacles at time t. The resulting Convex-QCQP formulation is (with variables P^t, q^t, r^t),

$$\begin{aligned} \textbf{minimize} \quad & f_1 + \alpha f_2 + \gamma f_3 \\ \textbf{subject to} \quad & {z_a^t}^T P^t z_a^t + {q^t}^T z_a^t + r^t \leq -1 \\ & {z_g^t}^T P^t z_g^t + {q^t}^T z_g^t + r^t \geq 0 \\ & {z_{o_i}^t}^T P^t z_{o_i}^t + {q^t}^T z_{o_i}^t + r^t \geq 1 \\ & i = \{1, ..., k\}; P^t \succeq I. \end{aligned} \tag{1}$$

Here $f_1 = ||{z_g^t}^T P^t z_g^t + {q^t}^T z_g^t + r^t||_1$, $f_2 = ||{z_a^t}^T P^t z_a^t + {q^t}^T z_a^t + r^t||_2^2$, $f_3 = \sum_{i=1}^k \Psi^t(z_{o_i}^t)$, $\alpha \in (0, 1]$, and $\gamma \in (0, 10^{-3}]$. As mentioned earlier, function f_3 results in affinity of resulting ellipsoid with the obstacles identified at time t; γ decides the magnitude of this affinity.

4.2 ECAN Path Planning: Navigation Inside Ellipsoid

After finding a feasible ellipsoid at time step t, a direction for navigation inside the ellipsoid needs to be found. The navigation direction z_n should take the agent away from the surrounding obstacles, while should also lead the agent towards the goal location. Function f_1 in convex-QCQP tries to align the resulting ellipsoid such that the principal axis points towards the goal location, while f_3 distorts this alignment by making the ellipsoid reactive to surrounding obstacles. This section now demonstrates how navigation direction is found using the ellipsoid's Eigenvectors and

Eigenvalues. If $\Psi^t(z_g^t) = 0$ at time-step t, then the navigation direction z_n, is simply a (unit) vector from z_a^t to z_g^t. However, if $\Psi^t(z_g^t) > 0$ then the navigation direction has to be decided. This direction is found by first solving a convex program, which is different for 2D and 3D navigation problems. Following sub-sections describe the method of finding a navigation direction when $\Psi^t(z_g^t) > 0$.

4.2.1 2D-ECAN: Avoiding Obstacles

In 2D case, two direction vectors are defined, the principal direction vector z_p and the obstacle vector z_o, which correspond to eigenvectors corresponding to $\lambda_{min}(P)$ and $\lambda_{max}(P)$ respectively. Due to the objective functions f_1 and f_2 in (1), vector z_p correlates with the x-axis of the agent's local coordinate system ζ_a^2, where x-axis of ζ_a^2 is the direction of motion of the agent at any time t. Vector z_o is at $\pm 90^0$ rotation with z_p. Lets assume that at time t there are certain number of obstacles surrounding the agent (and hence, ellipsoid Ψ^t). These obstacles are divided into two regions: those lying above x-axis and those lying below it in ζ_a^2. If number of obstacles lying above is greater than that of below, then z_o is at -90^0 w.r.t. z_p and $+90^0$ (measured anticlockwise) otherwise. Let z_p^u and z_o^u be unit vectors in direction z_p and z_o respectively. A vector $z_e \in R^2, ||z_e||_2 \leq 1$ is then found by solving,

$$\begin{aligned} &\textbf{minimize} && -(z_e)^T z_p^u - \beta log((z_e)^T z_o^u) \\ &\textbf{subject to} && ||z_e||_2 \leq 1, \end{aligned} \tag{2}$$

where $\beta \in (0, 1]$ decides the trade-off in bi-criterion convex optimization problem. Fig-3 (left) depicts z_p, z_o and z_e for the 2D-problem.

4.2.2 3D-ECAN: Avoiding Obstacles

Since $P \in S_{++}^3$, its Eigenvectors are mutually orthogonal, forming a new coordinate system. Principal direction vector z_p corresponds to the Eigenvector corresponding to $\lambda_{min}(P)$ and points towards z_g^t. The other two Eigenvectors are then rotated accordingly to form a right-hand coordinate system and they define the obstacle vectors z_o^1 and z_o^2. Assume that there are certain number of obstacles in the fov of the agent (and now they surround Ψ^t). Next, a vector (named collision-vector) is drawn from the center of ellipsoid to each of the obstacles and projection of these vectors is taken on z_o^1 and z_o^2. Fig-3 (right) depicts z_p, z_o^1 and z_o^2. Two parameters s_1^+ and s_2^+ are defined. If the number of collision-vector having negative projection on z_o^1 is greater than the number of collision-vector having positive projection on z_o^1, then $s_1^+ = +1$, and $s_1^+ = -1$ otherwise. In the same manner s_2^+ is computed, by taking projections on z_o^2. These two parameters, s_1^+ and s_2^+, effectively measure the number of obstacles in positive and negative z_o^1 and z_o^2 direction respectively. Let z_{pu}, z_{ou}^1 and z_{ou}^2 represent unit vectors in the direction z_p, z_o^1 and z_o^2 respectively. Also, let λ_1 and λ_2 are the Eigenvalues of P corresponding to Eigenvectors to which z_o^1 and z_o^2 correspond, respectively. The resulting convex program for finding a vector $z_e \in R^3, ||z_e||_2 \leq 1$ can be compactly represented as,

$$\begin{aligned} &\textbf{minimize} && -log\left(\frac{(z_e^T z_{ou}^1)^{\frac{1}{\lambda_1}} \times (z_e^T z_{ou}^2)^{\frac{1}{\lambda_2}}}{\exp\left\{\frac{-z_e^T z_{pu}}{\lambda_{min}(P)}\right\}}\right) \\ &\textbf{subject to} && ||z_e||_2 \leq 1. \end{aligned} \tag{3}$$

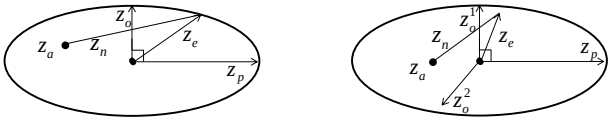


Figure 3: Left figure depicts inside ellipsoid vectors z_p, z_o, z_e and z_n for 2D-space and figure on right depicts z_p, z_o^1, z_o^2, z_e and z_n for 3D-space.

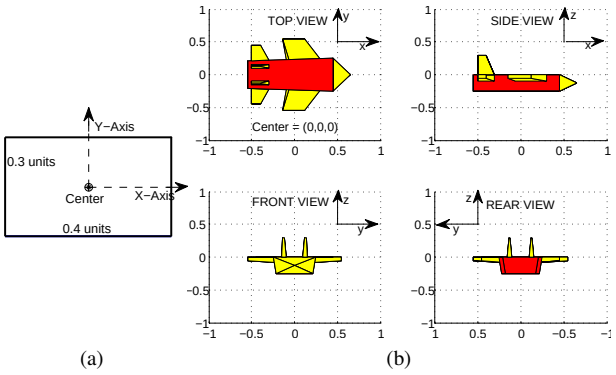
Here trade-off is decided by inverse eigenvalues (axes-length of ellipsoid) which resolves the directional movement. Directions playing lesser important role for navigation to the goal location are given less importance. This not only helps in efficient navigation, but also avoids unnecessary directions, which is important in 3D-planning [7].

4.2.3 Navigation Direction Inside 2D/3D-Ellipsoid

Once z_e is computed, a navigation direction z_n for moving inside the ellipsoid can be found, and the procedure is, roughly speaking, the same for 2D and 3D space. A positive scalar l_e is computed such that $z_b = (z_c + z_e \times l_e)$ is a point on the boundary of Ψ^t with z_c being its center. The navigation direction z_n is a vector from z_a^t to z_b, i.e. a vector from the current location of the agent inside the ellipsoid to the point z_b. Scalar l_e can be computed as,

$$\begin{aligned} l_e &= (-\Delta + \sqrt{\Delta^2 - 4\Lambda\Sigma})/(2\Lambda) \\ \Delta &= 2{z_c}^T P^t z_e + {z_e}^T q^t;\ \Lambda = {z_e}^T P^t z_e \\ \Sigma &= {z_c}^T P^t z_c + {z_c}^T q^t + r^t. \end{aligned} \tag{4}$$

Once z_n is computed, agent takes a small step l_n in this direction, such that it still remains inside the ellipsoid. When the agent is point size, $l_n = \min(\delta_1, \delta_2)$, where δ_1 is a predefined scalar, which may depend on speed of the agent or the step after which a re-computation of direction is required, or on the desired frequency of re-planning and $\delta_2 = ||z_a^t - z_b||_2$. However, when the agent has finite area/volume, then the length l_n is computed by solving a quadratic program (or equivalently a SOCP). Fig-2b shows an instance when the agent has finite area, and the length δ_2 is computed by solving the quadratic program (QP). Center of the agent is shown as red-dot in fig-2a. The figure was meant to show the importance of function f_2. Next section explains complete framework for the finite-area/volume agent and also the QP for computing l_n. Fig-3 left and right show vectors z_e (multiplied by l_e), and z_n for 2D and 3D space respectively.



(a) (b)

Figure 4: Finite area/volume agents used in the experiments: (a) shows a 2D-Agent with x-axis as the direction of motion; (b) shows a 3D-Agent with length of various parts described in the figure with the help of grid markings.

5 ECAN: Finite-Size Agent & Obstacle Architecture

This section extends the idea behind the point-agent model to the finite-area/volume agent avoiding the finite-area/volume obstacles. Section-5.2 explains the point-cloud representation of the finite-obstacles lying in fov of the agent. The finite agents, for 2D and 3D spaces, used in the experiments are shown in fig-4a and fig-4b respectively. Convex constraints in (1) kept the point-agent inside the resulting ellipsoid, same concept applies for the finite agent. Utilizing the property that a convex set $\mathbf{C}_1$ contains a set $\mathbf{C}_2$ (need not be convex), iff $\mathbf{C}_1$ contains the convex-hull of $\mathbf{C}_2$. Therefore the finite agents (where the 2D-agent forms a convex set while the 3D-agent forms a non-convex set) can be constrained to lie inside the ellipsoid by constraining the m-extremum locations on these agents to lie inside the ellipsoid Ψ^t. The extremum points for the 2D agents are the 4 corner points ($m = 4$), while for the 3D-agent (a plane), the extremum points are the corner points of the free end of front, rear and top wings (all are yellow in fig-4b); 4 corner points of the front and back ends of the central body (red in fig-4b); and the tip of the nose of the plane (font yellow colored tetrahedron in fig-4b), leading to a total of $m = 33$ locations on the plane. Thus the first constraint in convex-program (1) is replaced with the constraint ${z_a^t(i)}^T P^t z_a^t(i) + z_a^t(i)^T q^t + r^t \leq -1, i = 1, ..., m$; $z_a^t(i)$ denotes i^{th} location on the agent's body. Also, when the agent is finite-size, variable z_a^t represents its center at time t. Therefore, function f_2 in (1) now minimizes the distance of center of the agent with the boundary of resulting ellipsoid. The basic working principle of ECAN planning remains the same and only the step length l_n for moving inside the ellipsoid, in the direction z_n, changes. Step-length $l_n = \min(\delta_1, \delta_2)$, where δ_1 is a predefined scalar quantity, and δ_2 is found by solving a QP (next section) by constraining the agent to remain inside the ellipsoid Ψ^t.

5.1 Convex Quadratic Programming for Finding l_n

Let there be a data-structure D_a^t that stores location of different points on the agent's body in the form of vectors in agent's local axis system (x, y and z directions), as shown in fig-4 for both 2D and 3D agents, w.r.t to center of the agent (z_a^t). It also stores agent's local axis system according to the agent's orientation in space at time t. Therefore whatever be the orientation of the agent in space is, location of the points on the agent's body is given by $z_a^t(i) = z_a^t + D_a^t(i)$, where $D_a^t(i)$ is the vector (depends on the current orientation of the agent) for the i^{th} location on the agents body w.r.t the agent's center z_a^t. Therefore the variable δ_2 (and hence l_n), at time t, for moving inside the ellipsoid, in direction z_n, can be found by solving following Convex-QP (with variable x and δ_2):

$$\begin{aligned} &\textbf{maximize} \;\; \delta_2 \qquad (5)\\ &\textbf{subject to} \;\; x = \delta_2 z_n; \quad i = \{1, ..., m\}\\ &x^T P^t x \leq -(2P^t D_a^t(i)' + q^t)^T x + \Psi^t(D_a^t(i)') - 1. \end{aligned}$$

The constraints are obtained by expanding the ellipsoid constraint $\Psi^t(z_a^t + z_n \times \delta_2 + D_a^t(i)) <= -1$, for each of the m extremum locations and using $D_a^t(i)' = z_a^t + D_a^t(i)$. After solving the above QP, step-length can be found as $l_n = \min(\delta_1, \delta_2)$. Complete ECAN algorithm is explained as Algorithm-(1). Input parameters for subroutines are defined inside the curly braces, with optional input parameters further inside the square braces. Sub-routine *getObstaclesInFov* identifies obstacles in fov of the agent at time t; *getEllipsoid* forms the ellipsoid by solving-(1); *getDirectionInEllipsoid* finds direction z_e using optional input β (active) for 2D navigation by solving-(2) and by solving convex program-(3), without β, for 3D-navigation; *solveAnalyticEquation* finds l_e by using-(4); *getMotionDirection* finds z_n as mentioned in section-4.2.3; *isPointAgent* is a Boolean variable which is *true* for the point-agent and *false* otherwise; and *getLengthSolveQP* finds δ_2 by solving-(5).

5.2 Finite Area/Volume Obstacles

This section now incorporates finite obstacles for developing an architecture that handles finite-agent and finite-obstacles using convex formulation described in earlier sections. As described in sec-3, agent's fov is restricted by $(R_{fov}, \theta_{fov}, \phi_{fov})$. To incorporate finite obstacles, fov is discretized. The discretization parameters $(dr, d\theta, d\phi)$ represent discretization of respective components. The number of discretized grid-points is $N = (2\theta_{fov} + 1)(2\phi_{fov} + 1)R_{fov}/(dr.d\theta.d\phi)$ for 3D space and $N = (2\theta_{fov} + 1)R_{fov}/(dr.d\theta)$ for 2D space. This discretization is done in the agent's local coordinate system, and therefore needs to be done only once throughout the navigation. At any time t, these points are mapped to global coordinate system to get global coordinates of these discretized grid-points. At time step t, once the agent finds any finite-obstacle in its fov, the grid-points lying on the body of these obstacles are marked as point-obstacles in the environment. The

Algorithm 1 -Ellipsoidal Constrained Agent Navigation

- initialize parameters: $\delta_1, \gamma, \alpha, \beta$ and ϵ (small constact)
- initialize variables: $V_{fov} = (R_{\text{fov}}, \theta_{\text{fov}}, \phi_{\text{fov}}, dr, d\theta, d\phi)$
- initialize agent (at $t = 0$) and goal location: z_a^0, D_a^0, z_g

while $(||z_a^t - z_g||_2 > \epsilon)$
  $O^t \leftarrow$ *getObstaclesInFov*$\{z_a^t, D_a^t, V_{fov}\}$
  $\Psi^t \leftarrow$ *getEllipsoid*$\{z_a^t, O^t, z_g, D_a^t, \gamma, \alpha\}$
  if $(|\Psi^t(z_g)| > \epsilon)$
    $z_e \leftarrow$ *getDirectionInEllipsoid*$\{\Psi^t, D_a^t, O^t, [\beta]\}$
    $l_e \leftarrow$ *solveAnalyticEquation*$\{\Psi^t, z_e\}$
    $z_n \leftarrow$ *getMotionDirection*$\{z_e, l_e, \Psi^t\}$
    if (*isPointAgent*)
      $l_n \leftarrow \min(\delta_1, ||z_a^t - z_g||_2)$
    else
      $\delta_2 \leftarrow$ *getLengthSolveQP*$\{z_a^t, z_n, \Psi^t, D_a^t\}$
      $l_n \leftarrow \min(\delta_1, \delta_2)$
    endif
  else
    $z_n \leftarrow (z_g - z_a^t)/(||z_g - z_a^t||_2)$
    if (*isPointAgent*)
      $l_n \leftarrow \min(\delta_1, ||z_a^t - z_g||_2)$
    else
      $\delta_2 \leftarrow$ *getLengthSolveQP*$\{z_a^t, z_n, \Psi^t, D_a^t\}$
      $l_n \leftarrow \min(\delta_1, \delta_2)$
    endif
  endif
  $z_a^{t+1} \leftarrow z_a^t + z_n \times l_n$
  $t \leftarrow t + 1$
endwhile

getObstaclesInFov subroutine uses these discretization parameters to convert the part of the finite-obstacles lying in the agent's fov to a point-cloud. Also, this discretization is done only for simulation purposes. In the real-world implementation these points can be extracted from the output of any perception sensor, for example, coordinates of pixels defining the obstacles in case of image based perception, or using LADAR sensors ([8],[14]).

6 Experiments

This section lists various experiments to demonstrate performance of ECAN in cluttered environments, both using the point and finite obstacle. The purpose of including point obstacle is to increase the random clutterness in the environment. Finite agents used in the experiments for 2D and 3D navigation are shown in fig-4. Default values of parameters are: $(\alpha, \beta, \epsilon, R_{fov}, \phi_{fov}, d\phi) = (0.1, 1, 10^{-2}, 5, 40^0, 0.5^0)$; $\gamma = 5 \times 10^{-5}$ when only point obstacles are present, and $\gamma = 5 \times 10^{-4}$ otherwise; and $(\delta_1, \theta_{fov}, dr, d\theta) = (1, 80^0, 0.2, 1^0)$ & $(2, 40^0, 0.1, 0.5^0)$ for 2D & 3D navigation respectively. In all the experiments, environment is completely unknown and unseen by the agent, and also the agent has no knowledge about the environment beyond its field-of-view.

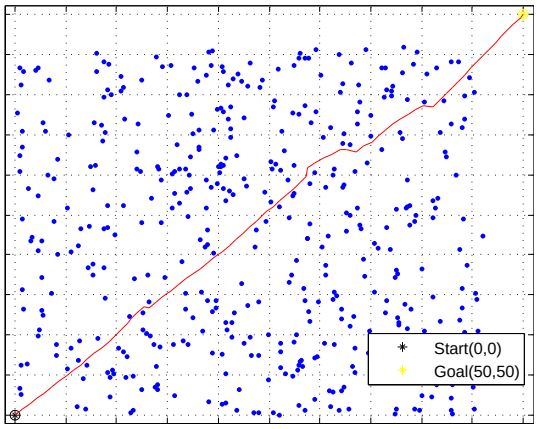


Figure 5: 2D-navigation: point obstacles with finite agent.

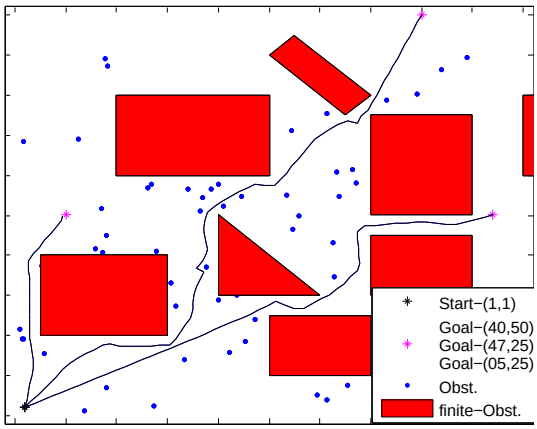


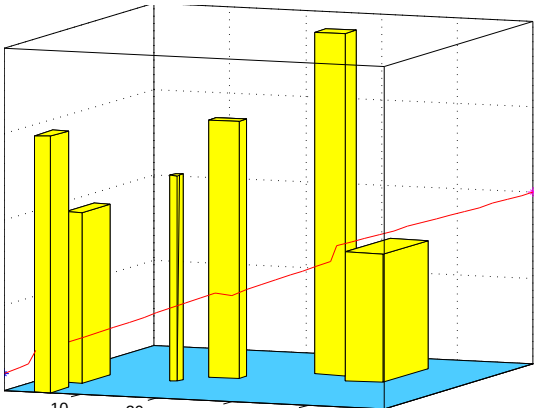
Figure 6: Three experiments for 2D-Navigation among finite and random-point obstacles with finite-agent.

6.1 2D-Navigation: Point Obstacels with Finite Agent

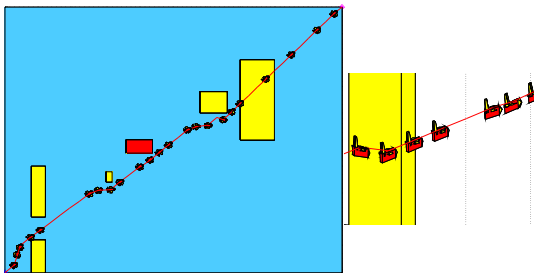
This experiment demonstrates ECAN's ability to plan a path and discover gaps, through which the finite-agent can pass, in randomly cluttered environment with point-mass obstacles, to reach the goal-location. Fig-5 shows a resulting path with randomly generated 446-point obstacles. At $t = 0$ agent is facing towards the goal location.

6.2 2D-Navigation: Finite-Agent & Mixed Obstacles

Three experiments shown in fig-6 demonstrate navigation with ECAN to 3-different locations with same initial conditions. These experiments demonstrate that the point obstacle representation of the boundary of finite obstacles is sufficient for ECAN to avoid any kind of finite obstacles. A spline can be fit ([21]), to further get a smooth path inside the ellipsoid (to avoid sudden turns), by considering the agent's currnet location inside the ellipsoid, ellipsoid's center and the point z_b on the boundary of ellipsoid.



(a)



(b) (c)

Figure 7: 3D-Navigation with finite agent; $z_a^0 = (0, 0, 1)$ and $z_g = (50, 50, 10)$.

6.3 3D-Navigation: Finite-Agent & Finite Obstacles

The 3D-agent (a fighter plane shaped UAV) navigates successfully in completely unknown and unseen environment using the finite-agent architecture of 3D-ECAN algorithm. To avoid clutterness, and to show clearly the path planned in online fashion, agent is not shown in fig-7a. Fig-7b depicts projection on (x, y)-plane and non-vertical turns to avoid obstacles. It shows the agent at key locations, such as when it is elevating and making turns. A magnified image of the agent is shown in fig-7c, while avoiding the red-top building (fig-7b). At $t = 0$ agent is facing at $(1, 1, 0.5)$.

6.4 Experimental Analysis of Computational Time

This section experimentally analyzes computation time required by the convex programs. ECAN was implemented on MATLAB-R2008a using non-commercial CVX ([22]) on Sony Vaio Laptop, 2.66-GHz Core i5-480M, 4-GB RAM, running Windows-7 64-bit. First, the case of 2D-finite obstacle with finite-agent is examined. These results are shown specifically for the path leading to (40,50) in fig-6 and conforms with the average result over several random 2D-domains. The total planning steps is 72, i.e. $t = \{0, ..., 71\}$. The average time taken by convex program-(1) is 0.15963-second and standard deviation 0.01997 with number of constraints varying from 6 to 130 including the

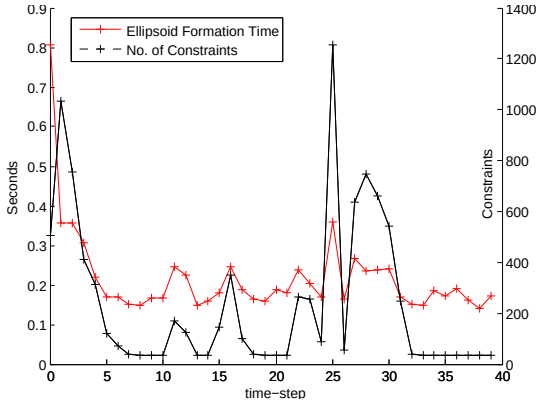


Figure 8: 3D-Navigation: Ellipsoid formation time and number of constraints at each time step t; $t = \{0, ..., 39\}$.

positive-definite constraint on P^t; average time taken by convex problem-(5) is 0.36788-second over 39 calls to the program; and the average time by problem-(4) is 0.0933-second over 72 calls. For 3D-path planning shown in fig-7a, time taken to form the ellipsoid at each time-step along with the number of constraints in the QCQP is shown in fig-8. The excessive time taken at $t = 0$ is due to the close proximity of the agent to ground (at $t = 0$ and hence point obstacles) and with a wall standing in front, with length and width of the agent taken into account, it restricts the ellipsoid by a great amount. For all other steps, it can be seen that the ellipsoid adjusts itself accordingly, thereby reducing the computation time. The average time taken for convex problem-(5), over 39 calls, to find δ_2 is 0.09097-second, with maximum time of 0.12082-second at $t = 0$; average time taken by convex program-(4) to find z_e is 0.4068-second and was called 12 times.

7 Discussion, Conclusion & Future Work.

This paper presented a Convex-QCQP based 3-stage convex-optimization algorithm ECAN for online path planning in completely unknown and unseen continuous environments. The algorithm has apparent computational advantage over integer-programming based path planning (see [11],[12] for comparison), and moreover is able to plan in unseen spaces, which most of the integer-program based path planning algorithms can't. However, a key point that is not addressed in this paper for 3D-planning is to determine the rotation, about local x-axis, of the 3D-agent according to the eigenvector-coordinate system of the ellipsoid. The 3D-ellipsoid not only provides a navigation direction, but also there's an important information about how the agent should rotate itself to nagivate as far as possible inside the ellipsoid. This can be particularly useful for fixed-wing UAV navigation (high-speed), as it provides UAV with the path and also with information about orientation to avoid obstacles. Also, dynamics of the vechile and minimum turn radius constraints were not addressed

while planning the path. While determining z_e, an extra constraint of maximum deviation from current motion direction can be added that takes into account the turn radius. Addressing these areas requires a separate full length paper and will be addressed in future.

We would also like to comment over *unseen spaces*, emphasized thoughout the paper. The term *unknown spaces* has been used widely in the literature, specifically in the research using artificial intelligence and some of the references mentioned in the related work. Most literature first propose a method to plan in unknwon spaces, but eventually they build a method that first either captures model of the environment or requires a path from the agent's initial location to the goal location, and then path planning occurs. This makes the environment (partially or completely) known. Therefore, we emphasized *unseen spaces* to clearly specify that the agent is unaware of upcoming geometrical-hindrances and obstacles as it navigates through the completely unknwon environment.

A key advantage and the reson behind success of ECAN in *unseen spaces* is that the orientation of ellipsoid intelligently incorporates the direction leading to a goal location. This provides robustness to the ECAN in finding a feasible path (if it exists) to the goal location, no matter how cluttered the space is. Also, convex formulation guarantees an optimal solution at each time-step and makes ECAN computationally tractable.

References

[1] S. Boyd and L. Vandenberghe, *Convex Optimization* (Cambridge University Press, 2004).

[2] S. Sharma, E.A. Kulczycki and A. Elfes, Trajectory Generation and Path Planning for Autonomous Aerobots, *ICRA Workshop on Robotics in Challenging and Hazardous Environments*, 2007.

[3] K. Yang and S. Sukkarieh, Planning Continuous Curvature Paths for UAVs Amongst Obstacles, *In Australasian Conference on Robotics and Automation*, 2008.

[4] A. Stentz, The focussed D* algorithm for real-time replanning, *In Int. Joint Conference on Artificial Intelligence*, pp. 16521659, 1995.

[5] S. Koenig and M. Likhachev, Fast Replanning for Navigation in Unknown Terrain, *IEEE Transactions on Robotics*, 21(3), 2005.

[6] D. Ferguson and A. Stentz, Field D*: An Interpolation-based Path Planner and Replanner, *In Int. Sysposium on Robotics Research*, 2005.

[7] J. Carsten, D. Ferguson and A. Stenz, 3D Field D: Impoved Path Planning and Replanning in Three Dimensions. *IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, 2006.

[8] D. Dolgov, S. Thrun, M. Montemerlo and J. Diebel, Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments, *Int. Journal of Robotics Research*, 2010.

[9] A. Richards, Y. Kuwata, and J. How, Experimental Demonstrations of Real-time MILP Control, *AIAA Guidance, Navigation, and Control Conference*, 2003.

[10] T. Schouwenaars, J. How and E. Feron, Receding Horizon Path Planning with Implicit Safety Guarantees, *In American Control Conference*, Boston, M.A., 2004.

[11] T.A. Ademoye, A. Davari, C.C. Caltello, S. Fan, and J. Fan, Path Planning via CPLEX Optimization, *Southeastern Symposium on Systems Theory*, University of New Orleans, March 16-18, 2008.

[12] M.P. Vitus, V. Pradeep, G.M. Hoffmann, S.L. Waslander and C.J. Tomlin, Tunnel-MILP: Path Planning with Sequential Convex Polytopes, *Proc. AIAA Guidance, Navigation and Control Conference*, 2008.

[13] Lars Blackmore, Robust Path Planning and Feedback Design under Stochastic Uncertainty, In *Proc. of AIAA Guidance, Navigation and Control Conference*, 2008.

[14] Nicolas Vandapel, James Kuffner, Omead Amidi, Planning 3-D Path Networks in Unstructured Environments, *Proc. Int. Conference on Robotics and Automation*, 2005.

[15] J.J. Kuffner, Efficient optimal search of euclidean-cost grids and lattices, *In IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, 2004.

[16] J. Miura, Support Vector Path Planning, *In IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, 2006.

[17] O. Hachour, A three-dimensional collision- free-path planning, *Int. Journal of Systems Applications, Engineering & Development*, 4(3), 2009.

[18] T.-K. Wang, Q. Dang and P.-Y. Pan, Path Planning Approach in Unknown Environment, *Int. Journal of Automation and Computing*, 4(3), 310-316, 2010.

[19] T. Schouwenaars, M. Valenti, E. Teron and J. How, Implementation and Flight Test Results of MILP-based UAV Guidance, *Proc. of the IEEE Aerospace Conference*, 2005.

[20] Y. Kuwata and J. How, Three Dimensional Receding Horizon Control for UAVs, *AIAA Guidance Navigation and Contro Conference and Exhibit*, 2004.

[21] T. Ersson and X. Hu, Path Planning and Navigation of Mobile Robots in Unknown Environments, *In IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, 2001.

[22] CVX: MATLAB software for disciplined convex porgramming by M. Grant and S. Boyd: http://cvxr.com/cvx/