ISSN 2007-9737

Maintaining Visibility of a Landmark using Optimal Sampling-based Path Planning

Rigoberto Lopez-Padilla[1], Rafael Murrieta-Cid[2]

[1] Centro de Innovación Aplicada en Tecnologías Competitivas (CIATEC), León, Mexico,

[2] Centro de Investigación en Matemáticas (CIMAT), Guanajuato, Mexico

rigolpz79@gmail.com, murrieta@cimat.mx

Abstract. An approach to extend sampling-based path planning algorithms to include visual restrictions is presented. This approach deals with visual constraints during the sampling and optimization processes. Four visual constraints are imposed during sampling: 1) keep the landmark within the sensor field of view, 2) avoid landmark occlusions, 3) maintaining landmark features near the image center, and 4) limit changes in landmark view orientation. These last two are imposed during path optimization. The robot task is to maintain these constraints, in an environment with obstacles, while the robot changes configurations. The sampling-based motion planning algorithm imposes and maintains both physical and visual restrictions. The process uses a collision checker to detect self- and obstacle-collisions, or landmark occlusions. To infer the landmark visibility, the algorithm dynamically builds a 3D model of camera field of view as seen from the moving robot. To maintaining the landmark features close to the image center, a distance parameter from the field of view boundary to the landmark is used and optimized. The camera roll angle was included as another element to be optimized, limiting changes in orientation. The algorithm has been implemented, and both results in simulation and experiments using a real robot manipulator are presented.

Keywords. Path planning, industrial robot, occlusion-free path, visual path.

1 Introduction

Maintaining fixed landmark visibility has been used in robotics to improve localization, navigation, object recognition, object manipulation, 3D reconstruction, quality inspection, etc. This task has been performed using motion planning, optimization, and visual-servoing techniques. Roboticists have recently focused on integrating these techniques [6, 7]. In our approach the motion planning, with visual constraints, maintains landmark visibility and provides good landmark visual acuity.

1.1 Related Work

This approach is related to techniques that search the robot state space to develop collision-free and occlusion-free paths for eye-in-hand robots. In [6] the authors call these techniques path planning for visual-servoing. They also divide these techniques into four groups: (1) Image space path planning, (2) Optimization based planning, (3) Potential Field-based path planning, and (4) Global path planning. The approach is related to motion planning algorithms that impose physical and visual constraints to build a collision- and occlusion-free path.

Image space motion planning creates a path for the camera and then verifies path feasibility in robot configuration space. In [7], for example, the authors present an approach that partitioned the visual-servoing problem into one employing several sub-targets, to simplify the control task, when the main target is far from the camera.

The algorithm developed is based on the Rapidly Exploring Random Tree (RRT) approach. They first sample the image space and project visual features in the image. If the image space restrictions are satisfied then the tree is

extended. The final camera trajectory is evaluated for configuration feasibility.

The principal disadvantage of image space motion planning is that suggested paths are not always feasible requiring re-planning until feasibility is obtained. To overcome this, some approaches have focused on planning in both image space and robot configuration space [11, 1] simultaneously. In [11] the authors presented a motion planning algorithm for visual-servoing based on the RRT approach. The algorithm built an exploration tree, to encode robot configurations and visual features and obtain useful paths preventing target visual occlusion during the visual-servoing process. While, simultaneously, satisfying joints limitations and field of view restrictions. In [1] the authors present an algorithm to build an exploration tree searching in both image space and configuration spaces.

This algorithm plans the robot movements allowing no robot base positional alterations. Approaches like those presented in [11] and [1] are limited to generate feasible robot trajectories without attempting to find optimal trajectories. Additionally, these approaches were computationally expensive as they detect targets in an image. Their sampling-based planning algorithms must search a virtual environment, requiring virtual camera images to perform image space searches adding significant computational time to first build and then analyze and process these images. The approach presented in this work could be potentially combined with robot navigation methods like the one presented in [10] to maintain visibility of an object while the robot navigates in the environment.

Optimization techniques have been attempted to obtain optimal trajectories with respect to cost. In one example, optimization is done by 'cost minimization' considering the error between the length of a certain "feasible" path and the length of the straight-line path between any impose restrictions.

The optimization required two steps: first the translation vector and then the rotation matrix were optimized [2]. The principal disadvantage of this type of approach is that it is limited to simple (sub-four jointed) robot systems and environments with limited numbers of obstacles [6].

Our approach can be used in complex environments and with robots having many degrees of freedom. The algorithm has been tested in a six degree of freedom (DOF) manipulator, equipped with a camera having a limited field of view. This robot/camera machine was mounted in an environment populated with obstacles. Any of these obstacles could produce a robot collision or could occlude the landmark. The algorithm computes a collision- and occlusion-free path between two configurations. The landmark must remain visible during the execution of the entire robot path. Furthermore, the algorithm is able to optimize feasible robot paths by iteratively re-planning paths during the overall path planning process.

1.2 Main Contributions

The main contributions of this work are the following: We extend the RRT* to maintaining landmark visibility in an environment with obstacles, considering both motion and visibility constraints. We model this problem to either (1) respect some constraints, or (2) reach optimization, and compare the results. Visual features are inferred using a collision detector to determine whether an object is in the camera field-of-view and is occluded or not. We develop a technique that infers object visual features using a collision detector for any robot configuration. Camera roll angle (γ_i) was restricted to angles that limit changes in the landmark view orientation. Metrics for the RRT* algorithm include to minimize robot trajectory length, maximize the distance between the landmark features and the image boundary, and to minimize camera roll angle changes. Each of these algorithmic improvements have been implemented in both simulation and experiments with a 6 DOF ABB robot.

2 Problem Formulation

The robot is equipped with a camera with a limited field of view considering width, height and range. It is assumed that this camera is placed on the robot end-effector. It is assumed that the workspace

is populated with obstacles and has a unique landmark (see Figure 1 (a)).

2.1 Landmark Visibility Problem

The main problem addressed here is to maintain consistent landmark visibility while computing a collision- and occlusion-free path between an initial and final robot configuration (see Figure 1 (b)). Requiring that the landmark remains in the camera field-of-view as the robot moves, remains un-occluded by objects, and motion occurs without physical robot-object or self-collisions, limits the number of paths that the robot could use as it moves in the environment.

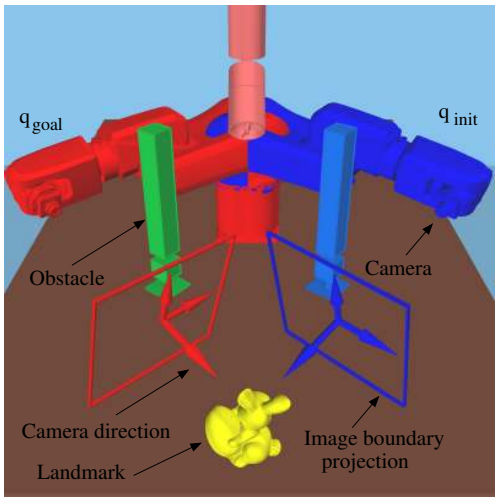
Let $\mathcal{C}$ denote the robot configuration space for moving in a 3D world and $\mathcal{V}$ be a space that indicates whether the landmark is/isn't contained in the camera field-of-view $v \in \mathcal{V}$ as it moves. For each robot configuration $q \in \mathcal{C}$ there is a scalar that indicates whether the landmark is fully visible in the camera's view. Let $\mathcal{C}_{obs}$ be the obstacle region where the robot will collide with obstacles or itself, and let $\mathcal{C}_{ocl}$ be a subset of $\mathcal{C}$ where the landmark is partially, or not, seen by the camera. Let $\mathcal{C}_{free}$ be free configuration space where the robot is collision-free, and the landmark occlusion free $(\mathcal{C} \setminus (\mathcal{C}_{obs} \cup \mathcal{C}_{ocl}))$. To build a planned solution, the algorithm searching for a path within $\mathcal{C}_{free}$ space is required. The planning problem is to find this feasible path such that:

— A path is a continuous function, $\tau : [0, 1] \rightarrow \mathcal{C}$.

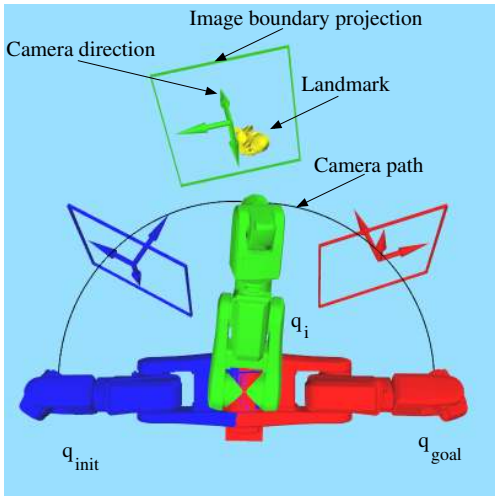
— A free path is a path in the free space $\tau \rightarrow \mathcal{C}_{free}$.

— A feasible path is a free path that starts at q_{init} and ends at $q_{end} \in \mathcal{C}_{goal}$

Robot configurations on the feasible path are subject to physical and visual constraints: the robot is not in self- or obstacle-collision and the landmark is completely un-occluded in the camera field-of-view.



(a)



(b)

Fig. 1. (a)Robot and environment, (b)This figure shows the concept of maintaining visibility of a landmark. The initial robot configuration q_{init} is represented with a blue robot (left) and the camera visibility at that configuration is represented with a blue frame (the image boundary projection). This frame is a slice of the camera field of view and the arrows represents the camera direction. The final robot configuration q_{goal} is represented with a red robot (right). An intermediate robot configuration is presented in green (middle). The desired camera path is shown as the black arc and every robot configuration to achieve this path has the landmark in the camera field of view

2.2 Landmark Visibility Constraint

A procedure was developed to assure no landmark occlusion that interfers landmark visibility. This procedure uses 3D models and a collision checker to infer whether the landmark is fully visible at a given robot configuration. Let $\mathcal{O}$ be a set of 3D models that represent all objects in the environment (including the robot itself at a given configuration – q_i). In order to determine the landmark visibility the procedure builds a 3D frustum model, that represents the limits of the camera field-of-view, and a Ray-casting model, that represents rays from the camera center to the landmark in the environment $o_l \in \mathcal{O}$.

2.2.1 Detecting Objects inside the Camera Field of View

Let μ_i be a 3D model of a frustum that represents the camera field of view region. The frustum height is the perpendicular distance between the planes indicating near focal length and far focal length of the camera. The other four planar surfaces represent the camera's imaging limits. This model is attached to the robot end effector (see Figure 2). Thus, an object o_l fully inside this frustum meets the visibility constraint if it is not occluded by another object o_h. Let $\mathcal{M}$ be the space that indicates whether the landmark is or is not contained in the frustum as the robot moves. A scalar $m_i \in \mathcal{M}$ is used to indicate this at a specific robot configuration q_i. To map a configuration q_i to a scalar m_i, a map is defined as $\mathbf{M} : \mathcal{C} \rightarrow \mathcal{M}$ or in functional notation $m_i = \mathbf{M}(q_i)$. Here $\mathbf{M}$ is based on the collision checker that detects any "collisions" between a solid 3D model of μ_i and the landmark model o_l.

2.2.2 Detecting Landmark Occlusions

Let κ_l be a 3D model that represents rays from the camera center to the landmark $o_l \in \mathcal{O}$ (see Figure 2). If the ray-casting model κ_l collides with an object model o_h (for any $h \neq l$) then the object o_h is in between the camera center and the landmark, o_l indicates landmark occlusion, regardless of whether o_l is in the camera's field view or not.

Let $\mathcal{K}$ be the space that indicates whether the ray-casting model collided with objects models for each $q_i \in C$. To map a configuration to a scalar k_i that indicates whether the ray-casting model κ_l collides with any object model o_h (for $h \neq l$) at that configuration, a map is defined as $\mathbf{K} : \mathcal{C} \rightarrow \mathcal{K}$. **K** uses a collision checker to detect occlusions.

This ray-casting model κ_l is dynamically modified while the robot moves, since the camera center pose changes with different robot configurations. The ray-casting model κ_l is constructed using the landmark model o_l and the camera center p. The landmark model o_l uses a triangle language (STL file format) for the 3D model representation, having j triangles. Each triangle has three line segments: $\overline{ab}$, $\overline{bc}$ and $\overline{ca}$. The ray-casting model κ_l is build by adding the camera center p to each line segment (in a triangle in o_l) to build three new triangles: Δabp, Δbcp and Δcap.

2.2.3 Detecting Full Landmark Visibility

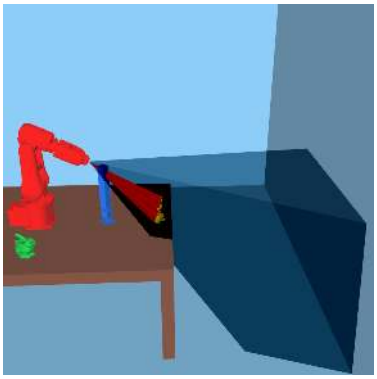
To infer landmark visibility at a given configuration q_i the procedure searches in $\mathcal{M}$ and $\mathcal{K}$. The landmark o_l is fully visible at a given robot configuration q_i if o_l is completely inside the frustum and o_l is not occluded.

Let $\mathcal{V}$ be a space that indicates whether the landmark $o_l \in \mathcal{O}$ is fully visible as the robot moves. v_i is a scalar that indicates landmark visibility, we define $\mathbf{V} : C \rightarrow \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{V}$ in functional notation $v_i = \mathbf{V}(\mathbf{M}(q_i), \mathbf{K}(q_i))$. The map **V** determines whether the landmark o_l is completely inside the frustum, and whether an object o_l is visible by the camera. Each v_i is a scalar that represents the landmark visibility. If the landmark $o_l \in \mathcal{O}$ is not fully visible v_i is zero.

In a feasible path τ_f, each $q_i \in \tau_f$ has a v_i equaling one, indicating that the landmark is fully (camera) visible.

(a) Not visible

(b) Visible



(c) Occluded

Fig. 2. Frustum model: camera frustum attached to the robot end effector. Ray-casting model: rays from the camera center to an object. (a) Landmark not visible by the camera: the Ray-casting model is not within the frustum model, (b) landmark visible by the camera: the Ray-casting model is within the frustum model, and (c) landmark occluded inside the camera field of view: the Ray-casting model collides with an object model

2.3 Modeling Landmark Visibility

Sampling-Based Path planning in the Joint-Image Space (as in [5]) could be computationally expensive, since the workspace-image projection process would be done many times in the sampling and other primitives of the planning algorithm. In contrast we present a planning approach in the joint space with visual constraints built into the algorithm. These visual landmark feature (position/orientation) constraints, in the algorithm, are inferred using only workspace information without an image. How these visual constraints are included is explained below.

2.4 Landmark Visual Features Constraints

There are two main constraints to be respected. One wants to keep the landmark 'far' from the image boundaries and one wants that the orientation angle of the image (camera roll angle), γ_i) to be close to a specific value. Landmark distance to the image boundaries d_l^s is constrained to be greater than a threshold distance d_t^s and landmark orientation angle θ_l^s to be within a range of angles $[\theta_a^s, \theta_b^s]$. Since absolute image space information is not available, workspace information is used to limit d_l^s and θ_l^s.

To limit d_l^s, the distance d_l from the landmark model o_l to the frustum model, a 'nonsolid' frustum (denoted μ_i) is used. For computing the distance between a pair of 3D models, a library for proximity query was used (the Proximity Query Package, PQP [8]). Let $\mathcal{D}$ be a space that indicates the distance between the landmark model $o_l \in \mathcal{O}$ and the frustum model μ_i at a given robot configuration q_i. To map a configuration to a scalar d_i that indicates the distance, we define $\mathbf{D} : C \rightarrow \mathcal{D}$. Here $\mathbf{D}$ is based on a function of the proximity query package and d_i is the computed distance between a nonsolid frustum and the landmark model o_l. In Figure 3 (a) distance d_i is presented as a line segment (in green). Note that the landmark is inside the frustum and that the distance is computed between the closest faces of the two models. In Figure 3 (b) this distance is projected in the camera image.

To limit θ_l^s, the rotation angles over the camera pitch, roll axes and 3D Rigid-body transformations

[9] are used. We assume that the camera X-axis is pointing at the landmark in a given robot configuration q_i, changes in the roll angle γ_i (rotation about the camera x-axis) causes the landmark feature image to rotate. The camera roll angle (γ_i) then must be held to limited range of angles $[\gamma_a, \gamma_b]$. Angle γ_i for a given configuration q_i is calculated using the rotation matrix of the robot camera model (Equation 1). We define $\mathbf{R} : C \rightarrow \Gamma$ to map a configuration to a scalar γ_i that indicating roll angle, $\mathbf{R}$ is given by Equation 1 and γ is given by Equation 2 and $\gamma_i \in \Gamma$.

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}, \tag{1}$$

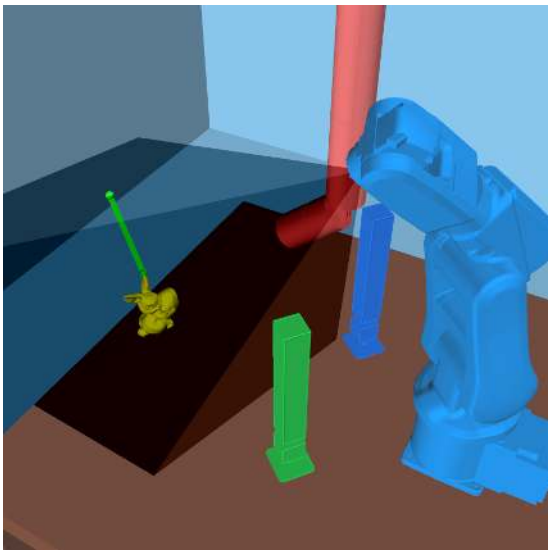
$$\gamma = |\text{atan2}(r_{32}, r_{33})|. \tag{2}$$

3 Path Planning to Maintain Landmark Visibility

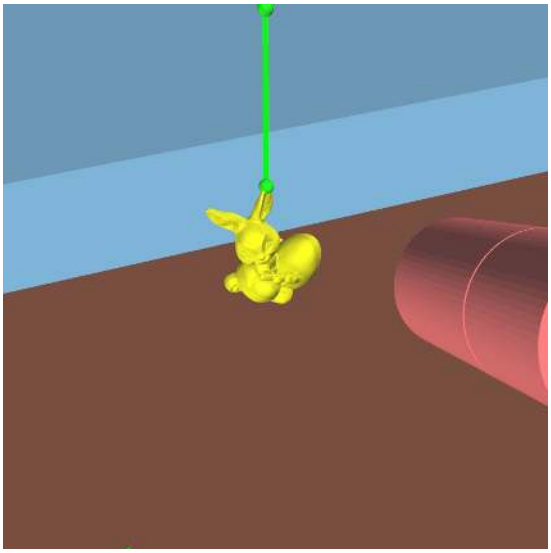
Sampling-based path planning algorithms can be extended to consider visual constraints by including them in the path planning algorithm. Here these visual constraints are used to extend the RRT* algorithm [3, 4]. The problem of constraining the position and orientation of the landmark visual features in the image space can be modeled as either a problem of respecting some constraints or an optimization problem. Below, we present both formulations and compare the solution results.

In this path planning problem the RRT* algorithm searches in C_{free} and uses information from $\mathcal{V}$ space, $\mathcal{D}$ space and Γ space in the algorithm primitives. In the original RRT* algorithm [3, 4] an exploration tree of robot trajectories is incrementally built. The algorithm starts from a root that represents the initial robot state.

At each iteration, a random sample from the free-state space is chosen and the tree is expanded by adding a new node to the tree for this random sample. Besides, a process can change the structure of the tree to reduce the length of the trajectories from the root to the leaves by choosing paths with closer nodes (using this strategy the authors proved asymptotic convergence to global optimality [3]).



(a)



(b)

Fig. 3. (a) Distance between landmark model and the frustum model d_i, (b) distance between landmark features and image boudaries d_l^s

To deal with the planning problem, some primitives were modified to include visual constraints, they are described below. We consider that the configuration space is equal to the state space, that is $\mathcal{X} = \mathcal{C}$.

3.1 Primitive Procedures

In the list, unmodified primitives procedures, from [3, 4], are summarized while modified primitives include more details.

— $\texttt{CollisionFree}(x_i)$ returns true (1) if the robot is not in collision with the environment (nor in auto-collision).

— $\texttt{LandmarkVisibility}(x_i)$ returns true (1) if the landmark is fully visible or false (0) otherwise. The visibility is computed using the map $v_i = \mathbf{V}(x_i)$ (see Section 2.2).

— $\texttt{SuitableFeatures}(x_i)$ returns true (1) if the landmark visual features fulfill the visual constraint (see Section 2.4) or false (0) otherwise. The function is based on the maps $d_i = \mathbf{D}(x_i)$ and $\gamma_i = \mathbf{R}(x_i)$ (see Section 2.2).

— $\texttt{StateValityCheker}(x_i)$ returns true if x_i satisfy the physical and visual constraints, i.e., the following sentence is true:

$$\texttt{CollisionFree}(x_i) \wedge \texttt{LandmarkVisible}(x_i) \wedge \texttt{SuitableFeatures}(x_i)$$

For implementation purposes we established a state validity checker function to be used with the OMPL [12]. OMPL itself does not include code for this checking, it was intentional, since defining validity depends on the type of problems to be solved [12]. Here we define a state validity check considering collision checking between loaded CAD models, landmark visibility, and feature suitability.

— **Sampling**: $\texttt{SampleFree}_i$ is a map, from random variables, to points in the free state space $\mathcal{X}_{free}$. Here a random state x_{rand} is determined to be in $\mathcal{X}_{free}$ using the $\texttt{StateValidityCheker}(x_{rand})$ function.

if x_{rand} is not in $(X)_{free}$ a new random configuration is evaluated until x_{rand} is in $\mathcal{X}_{free}$.

— **Distance**: Given two states: $x, y \in \mathcal{X}$, the $\texttt{Distance}$ function returns distance between the two states. The distance function is the L2 norm used for a State Space in OMPL.

— **Nearest Neighbor:** The function $\texttt{Nearest} : (G, x) \rightarrow v \in V$ returns the vertex in V that is "closest" to x in terms of the given distance function [3, 4].

— **Near Vertices**: The function $\texttt{Near} : (G, x, r) \rightarrow V' \subseteq V$ returns the vertices in V that are contained in a sphere of radius r centered at x [3, 4].

— **Steering**: The function $\texttt{Steer} : (x, y) \rightarrow z$ returns a point $z \in \mathcal{X}$ such that z is "closer" to y than x is [3, 4].

— $\texttt{Line}(x_1, x_2) : [0, s] \rightarrow \mathcal{X}$ denote the straight-line path from x_1 to x_2 [3, 4].

— $\texttt{c}(\sigma)$ is called the cost function, which assigns a strictly positive cost to all nontrivial collision-free paths [3, 4]. Using this approach the cost function is:

$$\texttt{c}(\texttt{Line}(x_{\text{current}}, x_{\text{new}})) = \texttt{Distance}(x_{\text{current}}, x_{\text{new}}) + \alpha_c \texttt{c}_c(x_{\text{new}}),$$

where α_c is a scaling factor, $\texttt{c}_c$ could be considered a clearance function imposing visibility constraints, $\texttt{c}_c(x_{\text{new}}) = \frac{1}{\mathbf{D}(q_{\text{new}})} + \mathbf{R}(q_{\text{new}})$, the distance inverse from landmark boundary to the frustum model boundary plus the γ_i. Using this cost function we look for maximizing the distance between the landmark boundary to the frustum boundary and minimize the (change in) γ_i.

— $\texttt{Cost} : V \rightarrow \mathcal{R}_{\geq 0}$ is a function that maps a vertex $v \in V$ to the cost of the unique path from the root of the tree to v. It is an additive cost function, so that $\texttt{Cost}(v) = \texttt{Cost}(\texttt{Parent}(v)) + \texttt{c}(\texttt{Line}(\texttt{Parent}(v), v))$. If $v_0 \in V$ is the root vertex of G, then $\texttt{Cost}(v_0) = 0$ [3, 4].

— `Parent` : $V \rightarrow V$ is a function that maps a vertex $v \in V$ to the unique vertex $u \in V$ such that $(u, v) \in E$. If $v_0 \in V$ is the root vertex of G, $\texttt{Parent}(v_0) = v_0$ [3, 4].

3.2 Algorithm 1: RRT* with a Landmark Visibility Constraints

We extended the algorithm RRT* from [3], to include visual constraints by modifying the `SampleFree`, `ObstacleFree`, `CollisionFree`, `Cost` and `c` primitives. The extended algorithm builds an exploration tree using the following steps ([3]):

3.2.1 Initialization (line 1, Algorithm 1)

The algorithm begins a search of the State Space by extending the tree starting at the root. The root depicts the initial robot state x_{init} and it is the first state of the path, e.i., $\tau(0) = x_{\text{init}}$. We have assumed that the initial and final robot states are in $\mathcal{X}_{free}$.

3.2.2 Sampling (line 3, Algorithm 1)

The sampling process rejects every state x that is not in $\mathcal{X}_{free}$. In our approach we use the `SampleFree` function not only to check for collision free states but to check visual feature requirements directly associated to the states. The $SampleFree$ function uses the `StateValidityCheck` function that was defined in OMPL by us.

3.2.3 Nearest Vertex (line 4, Algorithm 1)

The **Nearest** : $(G = (V, E), x)$ function returns the vertex in V that is "closest" to x in terms a L2 norm (`Distance`) over the angles of two configurations. The distance function does not take into account the visibility features properties because the state space is the configuration space $\mathcal{X} = \mathcal{C}$. Since a configuration q_i maps to v_i, d_i and γ_i the state space $\mathcal{X}$, here, is a subset of $\{\mathcal{C} \times \mathcal{V} \times \mathcal{D} \times \Gamma\}$.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{Obstacle\_Visual\_Free}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{RRT} * (\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
    $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + \texttt{c}(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do**
      if $\texttt{Collision\_Occlusion\_Free}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + \texttt{c}(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
        $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + \texttt{c}(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      end
    end
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do**
      if $\texttt{Collision\_Occlusion\_Free}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{new}}) + \texttt{c}(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$ **then**
        $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
      end
    end
  end
end
return $G = (V, E)$

Algorithm 1: RRT* with a landmark visibility constraints

3.2.4 Steering (line 5-6, Algorithm 1)

The function `Steer` returns a new state x_{new}, "closer" to x_{rand}, from $x_{nearest}$. The state x_{new} is an attempt to make a movement towards x_{rand}. The state $x_{nearest}$ and every node $v_i \in V$ fulfill the visual constraints, then x_{new} is also an attempt to maintain the visual constraints. To achieve this our `Obstacle_Visual_Free` function has extended the original `ObstacleFree` function in [3] and checks for both physical and visual constraints. If the landmark is fully visible and the landmark features are acceptable then an attempt to extend the tree towards x_{new} is made.

3.2.5 Extending the Tree (line 8-13, Algorithm 1)

Tests are performed to qualify the new vertex. A search for vertices near x_{new} is done to connect it along a minimum-cost path to G. The ratio r for the nearby vertices is defined as follows: $r = \min\{\gamma_{RRT} * (\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\}$ [3].

The `Collision_Occlusion_Free` function is an extension of the original `CollisionFree` function in [3]. This function evaluates the validity of motions between two specified states. In our implementation, we perform a discrete motion validation.

The `Collision_Occlusion_Free` function uses the `StateValidityChecker` function to check intermediate states along the path between any two states. The disadvantage of this discrete motion validation is that the motion is discretized to some resolution and states are checked for validity only at that resolution. If the resolution is too large, there may be invalid states along the motion path that escape detection. If the resolution is too fine, many states must be checked, significantly reducing planner performance [12].

The state x_{min} is chosen between nearby vertices $X_{\texttt{Near}}$ having the minimum-cost path to x_{new} and the edge $(x_{\text{min}}, x_{\text{new}})$ is added to E.

3.2.6 Rewiring the Tree (line 14-17, Algorithm 1)

The algorithm modifies the tree structure looking for optimal trajectories between the root and the leaves as it rewires the branches of the tree. This part of the algorithm is also modified by including the `Collision_Occlusion_Free` function.

Using the $\mathtt{c}$ cost function, landmark features properties can be optimized in terms of the visibility constraints imposed, that is $\mathtt{c}_c(x_{\text{new}}) = \frac{1}{\mathbf{D}(q_{\text{new}})} + \mathbf{R}(q_{\text{new}})$ which considers the 'closeness' of the visual features to the center of the field of view. The $\mathtt{c}_c$ cost can be set to zero if the user wishes to optimize the path length only, however the landmark most remain fully visible in any event.

3.2.7 Stopping Conditions

The path search can stop when the algorithm reaches n iterations or if a timed termination condition is reached. A feasible path is obtained if one or more vertices in $\mathcal{T}$ reach the goal region $\mathcal{X}_{goal}$. Here, any $x \in X_{goal}$ fulfills the visual constraints.

4 Results

We have tested our method using robot simulations and during experiments with a physical robot. OpenGL is used for visualization where all the objects in the environment are imported as 3D models using a triangle language. The PQP library is used to perform collision checking (and proximity query) and the Open Motion Planning Library (OMPL version 1.3.2) [12] is used to grow the exploration tree. Three different simulation experiments were performed to study some behavior differences in the RRT* algorithm with or without visual constraints. Physical experiments used a 6 DOF ABB IRB 120 industrial robot, which mounted a camera in its end effector, were also performed.

ISSN 2007-9737

4.1 Simulations

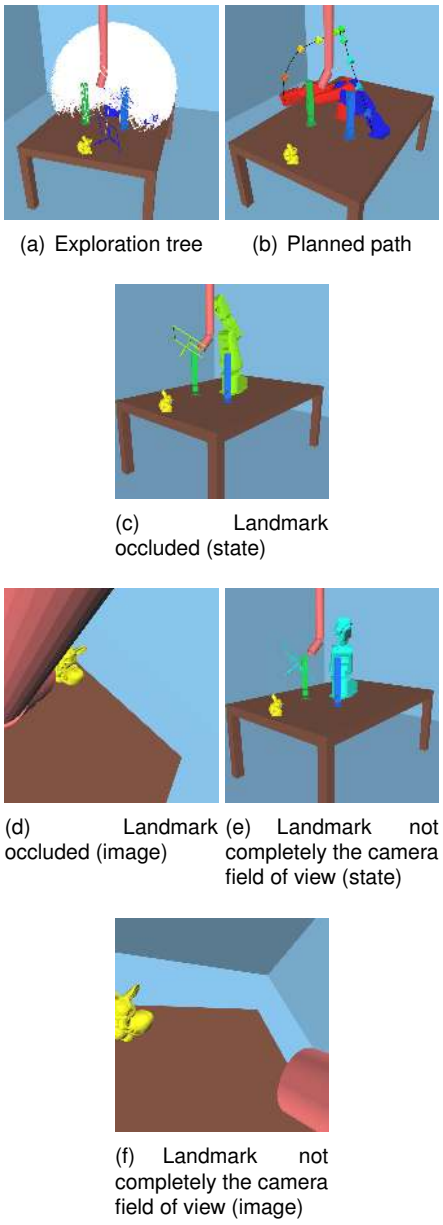
The initial and final robot configurations are the same for the three simulation studies. The environment includes several colored objects (the robot arm, miscellaneous workspace objects and the target landmark). To enhance visualization, different colors are used for each robot configuration, blue represents the initial robot configuration, red represents the final robot configuration, and other colors represent intermediate robot configurations along the planned trajectory. The landmark can be any of the objects on the table, but for these simulations the rabbit (yellow) is the target landmark.

Once an acceptable path is found, a representation of the exploration tree and the planned path were displayed. The number of iterations n is not constant, and a time termination condition to the building process was employed. The following settings were used in the algorithm: a) $k = 310$ neighbors, b) $r = 3.460682$ for the near function, c) the range of allowed roll angle γ_i is $[\pm 1.2]$ radians, and d) the minimum allowed distance d is 0.125 decimeters.

The 3D models used in the algorithm are different from those used in the visualization. In order to reduce the computational time, the number of triangles was reduced, without losing their spacial properties, having the same (or more) space in the environment. The number of triangles for all models in $\mathcal{O}$ is 1422. Each simulation was run 10 times using an dual-core PC processor, equipped with 12 GB of RAM, while running Linux. Table 1 presents data obtained from the three simulations.

4.1.1 Simulation 1: RRT* without Visual Constraints

The robot's task was to move the camera from the table's right to left side while avoiding collisions. This simulation was performed 10 times with a 60 seconds time limit. All ten simulation replays reached a solution within the time, see Table 1. Figure 4 shows the results of one simulation execution using the RRT* algorithm without visual restrictions. Figure 4 (a) displays a representation of the exploration tree were every node in the graph (in white) is the camera position at any state in the tree. Figure 4 (b) presents the planned path, each node in the path represents the robot camera and a segment line (in black) indicated the chosen camera positional transition between states.



(a) Exploration tree (b) Planned path

(c) Landmark occluded (state)

(d) Landmark occluded (image) (e) Landmark not completely the camera field of view (state)

(f) Landmark not completely the camera field of view (image)

Fig. 4. Simulation 1 with a RRT* without landmark visibility constraints

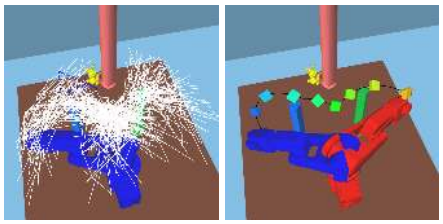
In this simulation, at times, the landmark is fully or partially occluded by an obstacle so is not completely visible by the camera. Figure 4 (c) shows a robot state where the landmark is occluded while Figure 4 (e) shows a robot state where the landmark is not completely inside the cameras field-of-view. This was anticipated since during this simulation the algorithm searches only for a path without obstacle collisions regardless of landmark visual quality.

4.1.2 Simulation 2: RRT* with Visual Constraints and Only Path Length Pptimization

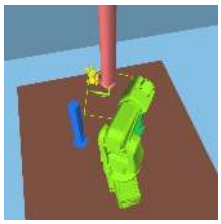
In the initial and final robot configurations, the physical and visual constraints are satisfied. In this simulation the robot's task was to avoid collision with the obstacles and to maintain the complete landmark camera visibility while the robot traverses the table's right side to its left side. The environment includes a 'lamp' that could easily occlude the landmark from many robot configurations (see Figure 5). This obstacle limits the occlusion-free robot motions since it is in between the landmark and the robot. Figure 5 (a) presents the exploration tree.

In this experiment there are fewer nodes than in the previous experiment (See Figure 4 (a) of the Simulation 1) since fewer configurations will meet the imposed visual constraints. To maintain landmark visibility, the algorithm found a feasible path, but the camera had to be rerouted below the bottom edges of the lamp to avoid collision and landmark occlusion (see Figure 5 (b)). As required, the landmark was always completely visible over the planned path. Additionally, landmark features are never too close to the image boundary and the landmark orientation is within the acceptable range. Figure 5 (d) shows a camera image, with the landmark close to the image boundary, as the robot moved over the path.

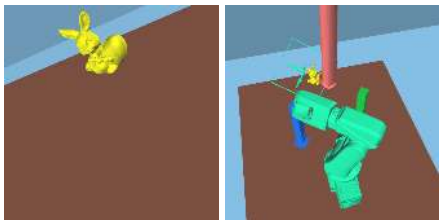
Note here, landmark features could be close to the image boundary since the visual constraints only guaranteed that a minimal distance from the boundary of the camera field-of-view frustum is assured. Figure 5 (f) shows a camera image indicating the maximum roll angle found over the planned path, which is within a range of valid angles. For this simulation, 10 replays were performed with a time limit of 300 seconds. A



(a) Exploration tree (b) Planned path



(c) Minimum d distance (state)



(d) Minimum d distance (image) (e) Maximum roll angle (state)

(f) Maximum roll angle (image)

Fig. 5. Simulation 2: (a) Exploration tree built using Algorithm 1, (b) the solution path to keep the landmark visible (only the camera path is displayed but the solution is a set of robot states where the landmark is always visible), (c) - (f) a robot configuration and its camera image in the solution path.

complete trajectory solution was found in 8 of 10 executions. Table 1 includes data obtained from the 10 simulation experiments.

4.1.3 Simulation 3: RRT* including Visual cost Optimization, the Landmark is as far as Possible from the Image Boundary and the Camera Roll Angle Close to Zero

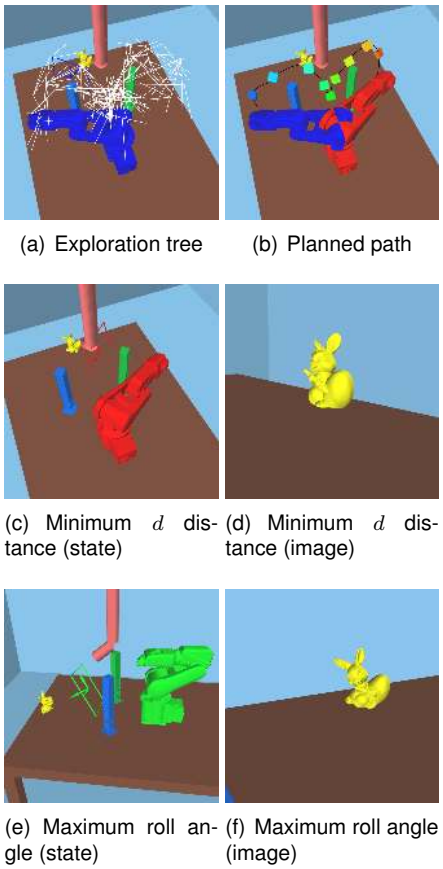
In this simulation, a landmark visualization optimization process was added. The optimization was designed to assure that the landmark features remained as far as possible from the image boundary and the landmark γ_i change was minimized. Thus, Here robot's task is avoid collision with the obstacles, avoid occlusion and keep the landmark (nearly) centered and 'upright' in the camera field-of-view while the robot moves from table's right to left side.

While Simulation 1 and 2 are included for comparison, this third simulation was wholly based on the newly developed optimal path planning approach we suggest. Figure 6 (a) presents the exploration tree. Since $\mathcal{C}_{free}$ is the same for the Simulation 2 and 3, the exploration tree can be seen to be similar to Simulation 2.

Figure 6 (b) presents the planned path. However, since the optimization process metrics had changed, the landmark features are closer to the center of the field-of-view and orientation is closer to the desired γ_i (zero) when compared to Simulation 2. Figure 6 (d) shows a close up image of the landmark as it would be seen across the planned path. The landmark is nearly centered (as required for optimality) in the image.

Figure 6 (f) shows the image at the maximum γ_i over the planned path, again, as required by the optimality constraints, it is nearly zero radians. For this simulation, 10 executions were performed with a time limit of 300 seconds. A complete trajectory solution was found in 10 of 10 executions. Table 1 includes data obtained from the 10 simulation experiments. We include a video of simulations 2 and 3 in the multimedia materials of this paper.

A video showing simulations 2 and 3 is also in the following link:
Link to the video:https://figshare.com/s/44616081306de618023d



(a) Exploration tree (b) Planned path

(c) Minimum d distance (state) (d) Minimum d distance (image)

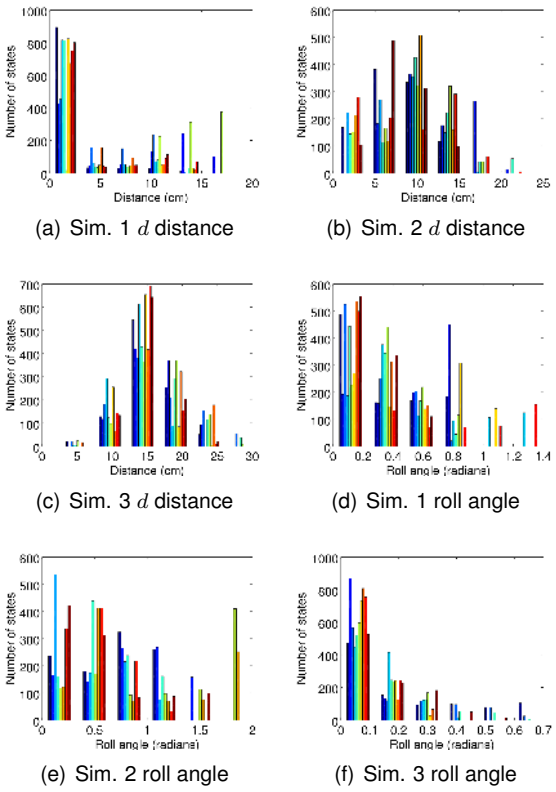
(e) Maximum roll angle (state) (f) Maximum roll angle (image)

Fig. 6. Simulation 3: (a) Exploration tree built using Algorithm 1, (b) the solution path to keep the landmark visible and away from the image boundaries), (c) - (d) the initial robot configuration and its camera image in the solution path

4.1.4 Analysis

In this section, an analysis of the simulation experiments is presented. We call the simulation runs an experimental set, thus 10 trajectories, to connect the initial configuration with the goal configuration, were developed, note that some runs may fail to reach the goal, so all run results can be averaged. In simulation 1, only controlled by collision avoidance, in simulation 2 both collision avoidance and visual constraints are maintained, with path length optimization. In simulation 3, the visual features cost is added to the optimization process.

ISSN 2007-9737



(a) Sim. 1 d distance (b) Sim. 2 d distance

(c) Sim. 3 d distance (d) Sim. 1 roll angle

(e) Sim. 2 roll angle (f) Sim. 3 roll angle

Fig. 7. Histograms of the simulation replays.(a)-(c) d distance, (d)-(f) roll angle

In our simulation notation, the distance d is the distance from the landmark to the field-of-view boundary. We consider an iteration one attempt to connect a new configuration to the tree. To obtain planned paths, we let the RRT* algorithm run for some prescribed time. In the case of Simulation experiment 1 we let the algorithm run for 1 minute, in the cases in which we satisfied visual constraints or optimize visual acuity (simulations 2 and 3), we let the algorithm run for 5 minutes, since many fewer potential configurations could satisfy the requirements.

We call first solution to the first path obtained by the algorithm that connects the initial and final configuration obtained within the time interval. Note that since the RRT* algorithm asymptotically optimizes the cost, the fist solution shall typically have a less good cost compared with the one obtained at the end of the time interval.

Table 1 (A. Simulation 1, B. Simulation 2, C. Simulation 3) contain data obtained from the simulations. Columns a) and b) shows the overall path length and path cost obtained after the RRT* was run over the entire time interval (60 seconds or 300 seconds for simulations 1, or 2 and 3, respectively). Column c) shows the number of vertices in the tree after the respective construction times, the number of vertices is smaller in simulations 2 and 3 since there are significantly fewer states that meet the visual constraints. Columns d) and e) shows the average of the d distance and the γ_i angle computed over the path configurations. Distances are larger and γ_i are smaller in simulation 3 because the optimization procedure maximizes the d distances (remembering this is the distance away from field-of-view boundary) and minimizes γ_i.

Columns g) to i) shows data for the first solution found at each simulation replay. Column g) shows first path cost, as expected it is greater than the path cost obtained at the end of the prescribed time reported in column b). Column i) shows the number of vertices of the tree, it is smaller than the number in column c). This is because the first solution is further optimized by the RRT* in the remaining time. Column h) shows the number of iterations; the number of iterations to find a first solution path increases in simulations 2 and 3 due to the visual constraints. However, the first and final solutions are found within the 300 seconds.

For each solution path, the d distance and γ_i were each used to create histograms for the simulations (see Figure 7). We compared the simulations histograms and noticed important behavior differences between them. Figure 7 (a), the histogram of d distance for simulation 1, displays many zeros since every time the landmark is not completely visible the d distance is set to zero. The distance trend, as expected, closes on zero appears since no attempt to force full visibility was enforced. Figure 7 (b), the d distance histogram of simulation 2 shows a tendency for keeping d above zero value, meaning that the landmark is always inside the frustum the same but higher d distances are observed in Figure 7 (c) for

simulation 3 since the optimization employed drove solutions with the landmark near the center of the frustum slice boundaries.

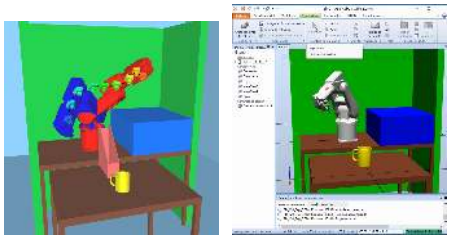
Figures 7 (d) and (e) shows the histograms for γ_i during simulation 1 and 2, notice the similarities. The histogram of γ_i, in Figure 7 (f) for simulation 3, clearly shows that γ_i is forced to be nearly zero. Thus, including a visibility (penalty) cost in the optimization procedure leads to a significant reduction in γ_i.

4.2 Real Environment

Our approach was tested is a real environment using an ABB IRB120 robot. The following settings were used in the algorithm: a) $k = 310$ neighbors, b) $r = 3.460682$ for the near function, c) the range of allowed roll angle γ_i is $[\pm 0.3]$ radians, and d) the minimum allowed distance d is 0.5 decimeters. The sum of triangles for all models in $\mathcal{O}$ was 408. Figure 8 (a) shows the simulated environment. We ran the planner 10 times and chose a solution path, this path is shown in Figure 8 (a).

Table 2 contains data obtained from the planner. In this table, the number of iterations in 300 seconds is greater than in Simulation 3; this is due to the smaller number of triangles used in this experiment. Figure 8 (b) shows the environment simulated in ABB's RobotStudio software; the planned path was successfully implemented and no collision was detected. Figure 8 (c) shows our laboratory, the computed path was executed in this lab using the ABB robot. Figures 8 (d)-(f) show eye-in-hand images captured during robot execution.

The paper multimedia material presents a video sequence captured by the camera mounted in the robot end effector, while the robot executed the planned path. These experiments demonstrate that our approach can be successfully implemented to obtain an optimal path using a real robot, provided that 3D models of the obstacles are available. A video showing the experiments in the real robot is also in the following link:
Link to the video:https://figshare.com/s/44616081306de618023d



(a) Planner environment (b) RobotStudio simulation

(c) Real environment

(d) Landmark

(e) Landmark

(f) Landmark

Fig. 8. Real environment test

Table 1. *Average for successful simulation replay (data was rounded to fit with the column format). + No solution found within 300 seconds

Legend: a) path length, b) path cost, c) number of vertices in the graph, d) average of the roll angle, e) average d distance (decimeters), f) number of iterations, g) initial solution path length, h) initial solution number of iterations, i) initial solution number of vertices in the graph.

No.	Data simulation 1. Tree construction time: 60 seconds								
	a)	b)	c)	d)	e)	f)	g)	h)	i)
1	7.5477	7.5477	22219	0.0902	0.3258	31454	15.61	16	8
2	7.1509	7.1509	22764	0.7290	0.5707	32124	15.56	74	42
3	7.3974	7.3974	23186	0.4697	0.2615	32724	13.23	35	24
4	7.3567	7.3567	22278	0.1482	0.5772	31453	14.37	20	14
5	7.2825	7.2825	22023	0.1729	0.2971	30973	19.07	61	44
6	7.5688	7.5688	22735	1.3700	0.4162	32063	13.56	13	10
7	7.6581	7.6581	22477	0.1594	0.5461	31559	9.8	212	127
8	7.4410	7.4410	22704	0.2378	0.2431	31989	16.21	39	22
9	7.3737	7.3737	22617	0.2542	0.4674	32029	23.11	46	26
10	7.4715	7.4715	22616	0.1726	0.2193	31772	18.19	43	22
Avg. *	7.4248	7.4248	22562	0.3804	0.3924	31814	15.87	56	34
No.	Data simulation 2. Tree construction time: 300 seconds								
	a)	b)	c)	d)	e)	f)	g)	h)	i)
1	10.6524	10.6524	732	0.7674	0.6635	182692	17.24	35581	62
2	11.5656	11.5656	667	1.2543	0.8583	192597	16.39	62806	89
3	10.7644	10.7644	663	0.7720	0.4064	184880	12.7	90750	219
4	10.6349	10.6349	546	1.0164	0.6379	166472	11.09	114531	319
5+	—	—	585	—	—	173070	—	—	—
6+	—	—	483	—	—	189391	—	—	—
7	11.0985	11.0985	664	0.9479	1.2133	195412	12.77	158143	475
8	11.0551	11.0551	691	0.8394	0.9365	221721	13.18	131034	298
9	10.3645	10.3645	654	0.8748	0.4753	207046	10.65	140441	350
10	10.5552	10.5552	646	0.7832	0.5133	198166	14.25	87829	151
Avg. *	10.8363	10.8363	658	0.9069	0.7131	193623	13.5338	102639	245
No.	Data simulation 3. Tree construction time: 300 seconds								
	a)	b)	c)	d)	e)	f)	g)	h)	i)
1	13.3081	136.6444	177	1.5365	0.2233	464135	175.43	273083	50
2	12.1779	97.7025	207	1.6387	0.0590	382945	176.62	103776	25
3	12.5840	121.5798	118	1.6573	0.1802	377586	190.55	258968	60
4	11.1268	113.9140	193	1.3177	0.1373	386441	179.45	142352	33
5	13.1672	119.3063	208	1.6610	0.1579	808397	216.11	568303	74
6	12.1210	107.3488	213	1.6865	0.1153	421746	226.45	82143	11
7	11.2181	107.9794	219	1.3503	0.0864	577084	171.97	198520	25
8	13.6679	113.0791	227	1.7191	0.0800	454258	190.07	113753	16
9	10.8290	95.9823	185	1.4427	0.0747	447962	102.27	297352	82
10	11.2491	107.5465	219	1.4734	0.1512	426150	176.65	126305	32
Avg. *	12.1449	112.1083	197	1.5483	0.1265	474670	180.56	216456	41

Table 2. * Average for successful planner replay (data was rounded to fit with the column format)

Legend: Real environment. Data are obtained from the planner replays: a) path length, b) path cost, c) number of vertices in the graph, d) average of the roll angle, e) average d distance (decimeters), f) number of iterations, g) initial solution path length, h) initial solution number of iterations, i) initial solution number of vertices in the graph.

No.	Data experiment. Tree construction time: 300 seconds								
	a)	b)	c)	d)	e)	f)	g)	h)	i)
1	5.4281	43.7794	469	0.0614	1.6668	6430283	71.37	608875	11
2	5.6538	42.3186	479	0.0516	1.8007	5857054	62.78	523254	18
3	5.6362	44.5623	474	0.0534	1.7392	6342627	76.3	185366	6
4	6.4968	43.7135	472	0.0664	2.1811	4964429	107.69	49948	4
5	6.1990	44.9021	526	0.0466	1.8027	6258154	87.74	234877	3
6	5.9428	47.1370	479	0.0617	1.7480	5445362	70.7	515437	5
7	5.8550	44.6246	481	0.0506	1.7358	4908820	73.4	203507	5
8	5.9621	42.9004	567	0.0659	2.0306	7509052	63.35	166999	4
9	5.8253	44.0943	508	0.0652	1.7688	6578829	54.07	245903	5
10	5.3815	43.8238	493	0.0320	1.6382	5927889	67.89	259264	7
Avg. *	5.838	44.1856	495	0.0555	1.8112	6022250	73.53	299343	7

5 Conclusions

We propose and implemented a variant of the RRT* algorithm to include landmark feature constraints (see Section 4.1.2). We further developed a RRT* variant that further constrained landmark features to be inside the camera field of view but closer to the image center. This was accomplished by including a distance metric optimization routine (see Section 4.1.3).

Finally, we built a roll angle γ_i optimizer to limit its range forcing the visual features orientation to remain as undisturbed as possible during path execution. The presence of obstacles increases the computational difficulty for robot moves in cluttered environments but we were able to solve the problem in reasonable time. We successful built planned trajectories that avoided obstacle- and self-collisions, optimized landmark observation (non-occluded and field-of-view centric), finding desired trajectories in reasonable processing time (in the order of some minutes using a standard desktop PC).

We presented solutions for environments represented with 1422 triangles (used in the algorithm for all models in $\mathcal{O}$) determined in under 300 seconds. We also present experiments that proved that our approach can be successfully implemented in a real robot, provided that the 3D environment models are known.

Only a few approaches had focused on motion planning for an eye-in-hand robot using visual constraints. Most of these approaches used visual features from an image to infer occlusion or to impose visual restrictions. In our approach, a collision checker is used to infer object visibility using workspace information. This is crucial for any efficient search algorithm in $\mathcal{X}$. In this work we proposed to use a collision checker to infer the objects visibility, a proximity query package to compute landmark distance from a field-of-view frustum boundary, and homogeneous transformations to computes the roll angle γ_i.

Like most motion planning approaches, the algorithm depends on the availability of 3D environmental models and it can be used in real robot applications when a reliable representation of the expected environment had been prepared.

During future studies, we propose to develop an algorithm to maintain visibility of several visual landmarks for operation of mobile-based manipulator robots.

References

1. **Bhargava, R., Mithun, P., Anurag, V., Hafez, A. A., & Shah, S. (2016).** Image space based path planning for reactionless manipulation of redundant space robot. *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, IEEE, pp. 4362–4368.
2. **Hafez, A. H. A., Nelakanti, A. K., & Jawahar, C. (2015).** Path planning for visual servoing and navigation using convex optimization. *Int. Journal of Robotics and Automation*, Vol. 30, No. 3.
3. **Karaman, S. & Frazzoli, E. (2011).** Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, Vol. 30, No. 7, pp. 846–894.
4. **Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., & Teller, S. (2011).** Anytime motion planning using the rrt. *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, pp. 1478–1483.
5. **Kazemi, M., Gupta, K., & Mehrandezh, M. (2009).** Global path planning for robust visual servoing in complex environments. *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 326–332.
6. **Kazemi, M., Gupta, K., & Mehrandezh, M. (2010).** Path-planning for visual servoing: A review and issues. In **Chesi, G. & Hashimoto, K.**, editors, *Visual Servoing via Advanced Numerical Methods*, volume 401 of *Lecture Notes in Control and Information Sciences*. Springer London, pp. 189–207.
7. **Kazemi, M., Mehrandezh, M., & Gupta, K. (2011).** Kinodynamic planning for visual servoing. *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2478–2484.
8. **Larsen, E., Gottschalk, S., Lin, M. C., & Manocha, D. (1999).** Fast proximity queries with swept sphere volumes. Technical report, Technical Report TR99-018, Department of Computer Science, University of North Carolina.
9. **LaValle, S. M. (2006).** *Planning algorithms*. Cambridge university press.
10. **Saldaña-González, G., Sánchez, J. C., Díaz, M. B., & Ata-Pérez, A. (2018).** Vision system for the navigation of a mobile robot. *Computación y Sistemas*, Vol. 22, No. 1, pp. 301–308.
11. **Shademan, A. & Jagersand, M. (2012).** Robust sampling-based planning for uncalibrated visual servoing. *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 2663–2669.
12. **Şucan, I. A., Moll, M., & Kavraki, L. E. (2012).** The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, Vol. 19, No. 4, pp. 72–82. `http://ompl.kavrakilab.org`.

Article received on 13/07/2018; accepted on 13/06/2019. Corresponding author is Rafael Murrieta-Cid.