Journal of Astronomical Telescopes, Instruments, and Systems

AstronomicalTelescopes.SPIEDigitalLibrary.org

Roadmap search based motion planning for MIRADAS probe arms

Josep Sabater
Santiago Torres
Francisco Garzón
José M. Gómez

Josep Sabater, Santiago Torres, Francisco Garzón, José M. Gómez, "Roadmap search based motion planning for MIRADAS probe arms," *J. Astron. Telesc. Instrum. Syst.* **4**(3), 034001 (2018), doi: 10.1117/1.JATIS.4.3.034001.

Roadmap search based motion planning for MIRADAS probe arms

Josep Sabater,[a,b,*] **Santiago Torres,**[b] **Francisco Garzón,**[c,d] **and José M. Gómez**[e]
[a]Institut d'Estudis Espacials de Catalunya (IEEC), Barcelona, Spain
[b]Universidad de La Laguna, Departamento de Ingeniería Informática y de Sistemas, La Laguna, S/C Tenerife, Spain
[c]Instituto de Astrofísica de Canarias (IAC), La Laguna, S/C Tenerife, Spain
[d]Universidad de La Laguna, Departamento de Astrofísica, La Laguna, S/C Tenerife, Spain
[e]Universitat de Barcelona, Departament d'Enginyeria Electrònica i Biomèdica, Barcelona, Spain

Abstract. MIRADAS is a near-infrared multiobject echelle spectrograph operating at spectral resolution $R = 20{,}000$ over the 1 to 2.5 μm bandpass for Gran Telescopio Canarias. It possesses a multiplexing system with 12 cryogenic robotic probe arms, each capable of independently selecting a user-defined target in the instrument field of view. The arms are distributed around a circular bench, becoming a very packed workspace when all of them are in simultaneous operation. Therefore, their motions have to be carefully coordinated. We propose here a motion planning method for the MIRADAS probe arms. Our offline algorithm relies on roadmaps comprising alternative paths, which are discretized in a state-time space. The determination of collision-free trajectories in such space is achieved by means of a graph-search technique. The approach considers the constraints imposed by the particular architecture of the probe arms as well as the limitations of the commercial off-the-shelf motor controllers used in the mechanical design. We test our solution with real science targets and a typical MIRADAS scenario presenting some instances of the two identified collision conflicts that can arise between any pair of probe arms. Experiments show the method is versatile enough to compute trajectories fulfilling the requirements. *© 2018 Society of Photo-Optical Instrumentation Engineers (SPIE)* [DOI: 10.1117/1.JATIS.4.3.034001]

Keywords: multiple object; spectrograph; probe arm; motion planning; collision-free; trajectory.

Paper 18010P received Feb. 26, 2018; accepted for publication Jun. 29, 2018; published online Jul. 24, 2018.

1 Introduction

Multiobject spectroscopy lies at the heart of modern astronomical instrumentation, in particular, in large telescopes, where the high oversubscription factor makes it mandatory to increase the observing efficiency of every instrument. This requirement for efficiency is even more demanding in instruments with long-lasting average integration times, like medium to high-resolution spectrometers. Hence, multiplexing systems (MXS) of different types form a part of every spectrograph currently being built for big telescopes. The inclusion of a multiplexer in an instrument involves, in most cases, adding complex optomechanical devices, which in turn, need to be commanded in a coordinated way to achieve the desired functionality.

MIRADAS[1] is a near-infrared multiobject echelle spectrograph operating at spectral resolution $R = 20{,}000$ over the 1 to 2.5 μm bandpass. The instrument, particularly designed for Gran Telescopio Canarias (GTC), is expected to enter its commissioning phase in late 2018. One of its remarkable features is a built-in MXS with 12 deployable and independently controlled probe arms with pickoff mirror optics, each feeding a 3.7×0.4 arc sec field-of-view to the spectrograph. Each probe field is transformed into three end-to-end slices of a fixed 3.7×0.4 arc sec format by a "slit slicer" present at the spectrograph input optics. This approach has the advantage of providing minimal slit losses in any astronomical seeing conditions better than 1.2 arc sec and, simultaneously, some limited two-dimensional spatial resolution. Besides, the spectrograph optics supports a range of configurations, providing a high degree of versatility. Depending on the needs of the science program, the observer will be able to choose between maximal multiplex advantage and maximal wavelength coverage, with several in-between options. Figure 1 shows a general view of the MXS bench and its probes.

Each of these robotic mechanisms patrols a limited region of a field of 5 arc min in diameter (about 250 mm.). The arm's pickoff mirror placed near the telescope focal plane relays light down the probe arm, where there is a collimating doublet lens, through a series of folds inside the mechanism. These folds are carefully designed to keep always a fixed optical path length while the arm is moved to variable target locations in the field of regard. Additionally, a closed loop structure provides the articulated mechanism with a high degree of stability when it works upside down. However, the kinematic behavior of the arm is not intuitive, which makes its control more difficult.

To observe 12 distinct user-defined targets at once, the arms need to be placed in the correct locations. As the tips of the articulated mechanisms approach the center of the MXS bench, see Fig. 2, room for maneuvering becomes scarcer and scarcer. Moreover, to make efficient use of the telescope's observation time, the MXS system requires a control unit capable of simultaneously moving all probe arms while, at the same time, avoiding collisions between each other. Therefore, in such a cluttered environment, it is vital to implement a piece of software computing and safely coordinating the trajectories of the arms.

*Address all correspondence to: Josep Sabater, E-mail: jsabaterm@el.ub.edu

2329-4124/2018/$25.00 © 2018 SPIE

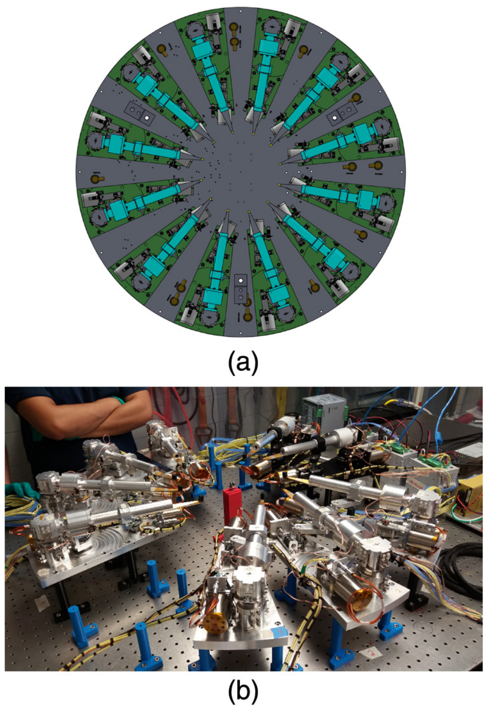


Fig. 1 The MIRADAS MXS system bench. (a) A drawing of the bench with 12 probe arms. The primary task of these mechanisms is to relay light from a given target located in the instrument's field-of-view to the spectrograph. The field-of-view is a circular area (250 mm) placed in the middle of the bench. (b) Nine real MXS probe arms at the lab ready for testing. They were arranged as if they were in the MXS bench.

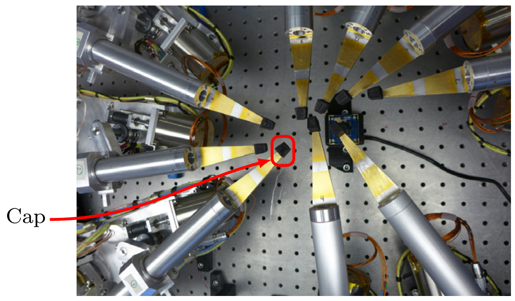


Fig. 2 Nine MXS probe arms at their final locations as delivered by the target allocation algorithm. The pickoff mirrors were covered by a black rubber cap during testing.

In this paper, we present part of the internals of the MXS control system software. Specifically, the algorithm responsible for generating collision-free motions for MIRADAS probe arms. The component of the control software related to the selection of the targets to be observed is not part of this work, as it will be further commented in Sec. 2. The algorithm described in this paper is executed once a given set of targets, appropriate for the physical constraints of the MXS system, has been identified.

Our motion planner is based on an iterative solution that individually determines the trajectories of each arm, verifying its feasibility with the help of a collision checking routine. The algorithm uses roadmaps containing several alternative paths, increasing this way, the arms maneuverability. They are constructed and systematically explored with different time profiles. These times profiles are achieved employing a discrete state-time graph, whose connections are dynamically built while a search is performed by a depth-first technique. This strategy intrinsically guides the search toward individual trajectories arriving at their destinations as soon as possible.

The rest of the paper is organized as follows. We begin by describing many of the main tasks performed by the MXS control system in Sec. 2. Next, in Sec. 3, we provide a brief introduction of the MIRADAS MXS probe arm because a minimum knowledge of this mechanism is required to understand the work presented here. We then, in Sec. 4, give a formal definition of the problem. The global approach of the prioritized solution we propose is discussed in Sec. 5. It fundamentally relies on two building blocks: an algorithm able to find a trajectory in a previously known dynamic environment and a prioritization scheme. The former is introduced in Sec. 6 and the latter in Sec. 7. Finally, in Sec. 8, we demonstrate the performance of our approach and conclude in Sec. 9 with discussion and extensions.

2 MXS Control System

The goal of the MIRADAS MXS control system is to place the pickoff mirror of each MXS probe arm in a particular target previously defined by a scientist. Although simple in concept, to accomplish this task, the system relies on several fundamental software packages appropriately connected. We can basically distinguish two different kinds of operations: offline and online. The first refers to planning operations executed well in advance prior to observing time. The offline operations are carried out by a piece of high-level software run in a desktop machine. The second group contains those operations commanding the instrument, which are typically run at night during observation. These operations are performed by real-time low-level control software interacting with the real hardware. Among these operations, there is one that is responsible for the appropriate execution of the arms motions previously planned. Note that only the offline motion planner is within the scope of this paper. The interested reader is referred to previous work of the authors for further details about the target allocator.[2]

2.1 *Offline Operations*

Each target comes determined by three different components: a sky position, a user assigned observing-priority, showing the importance of the given target in the overall observing program, and a unique alphanumerical label. There is no limitation on the values a user can assign to the target observing-priority field. Therefore, if convenient, scientists can employ elaborated observation-priority policies, including assigning a different value to each target.

Collision-free trajectories for a given set of targets are determined sequentially executing two different processing steps: (i) target allocation and (ii) motion planning. The first is essentially an assignment problem similar to those found in classical combinatorial optimization.[3] The aim in MIRADAS is to compute N pairs <arm, target> maximizing the total aggregated observing-priority, where N is the number of arms present in the system. The problem is, in addition, subjected to a set of constraints. As these constraints, as well as the objective function, can be modeled as linear expressions, the target allocation problem in MIRADAS is solved by means of a constrained integer linear program.[2] In two different situations, the allocator can automatically decide to position one or more arms at blank sky. First, if the MXS system has some arms not working and their real locations are known in advance. Second, if during computation, for any reason, a working arm cannot be assigned to a target, and then the allocator will position the arm at a predefined park position. In Fig. 2, a real assignment plan for nine arms delivered by our target allocator is shown.

Once an assignment plan is successfully determined, the motion planning algorithm computes collision-free trajectories so that each arm safely reaches its assigned target. As MIRADAS has been conceived as a GTC common-user instrument for many years to come, the target allocation and the motion planning tasks are totally decoupled. The motion planner makes no distinctions among the targets. Its goal is to find trajectories for all of them without taking into account their observation-priority. Furthermore, the motion planner will deliver these trajectories expressed in an actuator independent format.

The planner proposed follows an arm-prioritized approach, widely used in robotics due to its simplicity and ability to scale well in real environments. Priority in this context, as we will see in Sec. 5.2, refers to the order in which the arm trajectories are computed. Each arm is given a unique priority value and the algorithm proceeds sequentially from the highest priority arm to the lowest one. At each iteration, an arm determines its trajectory from its current position to the target assigned such that it avoids colliding with the higher-priority arms, whose trajectories were planned in previous iterations. In principle, any priority scheme can be used; however, we initially employ a fixed priority policy favorable to those arms showing an estimated larger arrival time.

Although it is beyond the scope of this work, we mention that if the motion planner fails to compute trajectories, then replanning is performed with a different arm-priority scheme until it succeeds or a predefined number of schemes have been tried. If no motion plan is found, problematic targets are temporarily discarded and a different assignment plan is determined. Note that all this procedure happens well before the observations, so there is no a priori impact on the observation efficiency.

To guarantee safe trajectories for the real world, strong synchronization among the different arm motions is required. The planner uses a function to estimate possible collisions. It takes into account a small safety area around each arm to prevent issues caused by small timing divergences. This area also covers differences between arms due to manufacturing tolerances.

Although many journals and conferences have proliferated in robotics about motion planning, in astronomical instrumentation, it is an incipient field or research. Lately, some works focusing on how trajectories are generated for several positioners have appeared. In infrared imaging spectrograph, a spectrograph equipped with three probe arms, a navigation function relying on potential fields is utilized.[4] This classic technique was previously applied for fiber positioners.[5] Although the idea behind is very attractive for its simplicity, in some particular environments, dead locks might arise.[6] Finally, TAIPAN, a fiber-based multiobject spectrograph, presents several distinct motion planning algorithm for its positioners.[7] Among them, like in MIRADAS, one depends on a graph search technique.

Conceptually, MIRADAS has many similarities with KMOS, a multiobject spectrograph being used at the ESO's very large telescope.[8,9] Both instruments employ deployable integral field units in the form of probe arms distributed around a circular bench to observe several user-defined targets simultaneously. However, technically, they are different. First, the mechanical design of the KMOS probe arms makes the task of positioning the arm in a given focal plane location more intuitive than in MIRADAS. The KMOS mechanism presents a polar (r, φ) motion approach, whereas in MIRADAS, as we will see in Sec. 3, the structure of the arm is more complex, resulting in more elaborated motions. Finally, KMOS observations can be prioritized, ranging from priority 1 to 3, whereas in MIRADAS, the values that the user can assign to the observation-priority field is not restricted to a given set of values. This is a consequence of the target allocation and the motion planning steps being decoupled in MIRADAS, as said earlier, that seems not to be the case for KMOS.

2.2 *Online Operations*

As stated earlier, the motion planner returns the sequences of actions the arms have to perform in a format totally independent of the real hardware used to move the arms. The computed sequences are delivered to an additional low-level software, whose implementation is specific to the particular motors and controllers actuating the arms. This lightweight layer translates the hardware-independent motions computed by the planner into controller commands, communicates with the motor controllers, and manages synchronization.

3 MXS Probe Arm

The arm, as shown in Fig. 3, is a two degree-of-freedom closed-loop mechanism composed of four joints (J_1, J_2, J_3, and J_4) and three links, providing to the whole structure a planar motion. These links are mainly tubes containing optical components to relay a beam of light from the pickoff mirror [Fig. 3(d)] to the spectrograph. The location of any element of the arm, including the tip mirror, can be determined by the rotation angles, denoted by θ_1 and θ_2, of the revolute joints J_1 and J_2, respectively. While the latter is unconstrained, being able to move in the interval $[0,2\pi)$, the motion of the former is constrained to the range $[0,\pi]$. Finally, the motion of J_4, which freely slides over the arm's longest tube, is constrained by the cylindrical cabinet containing the collimator.

For more information about accuracy and demonstration of an MXS probe arm, refer to previous work.[10]

As it will be discussed in Sec. 4, although the articulated mechanism moves in the MXS bench, its trajectories are determined in a distinct world, in its configuration space (C-space). Since its introduction in the field of robotics in the early 1980s,[11] this space has been widely used to develop motion planning techniques. Its main advantage is that the volume occupied by the mechanism in its workspace becomes a single point in the C-space. In our case, the C-space consists of all arm configurations $q_i = (\theta_{1_i}, \theta_{2_i})$; see Fig. 4. Two different subsets of configurations can be distinguished there: (i) those which are feasible and (ii) those which must be avoided; see Sec. 3.1 for more details. Therefore, during motion planning, those potential trajectories traversing prohibited areas of the C-space have to be immediately discarded.

3.1 *Transformations*

The transformations between both spaces can be carried out with the particular solutions to the forward and inverse kinematic problems.[12] As a result of the complex structure of the mechanism, the inverse problem is more conveniently solved by geometric methods. However, considering the linkage model in Fig. 5, an expression can be formulated for forward kinematics as follows:

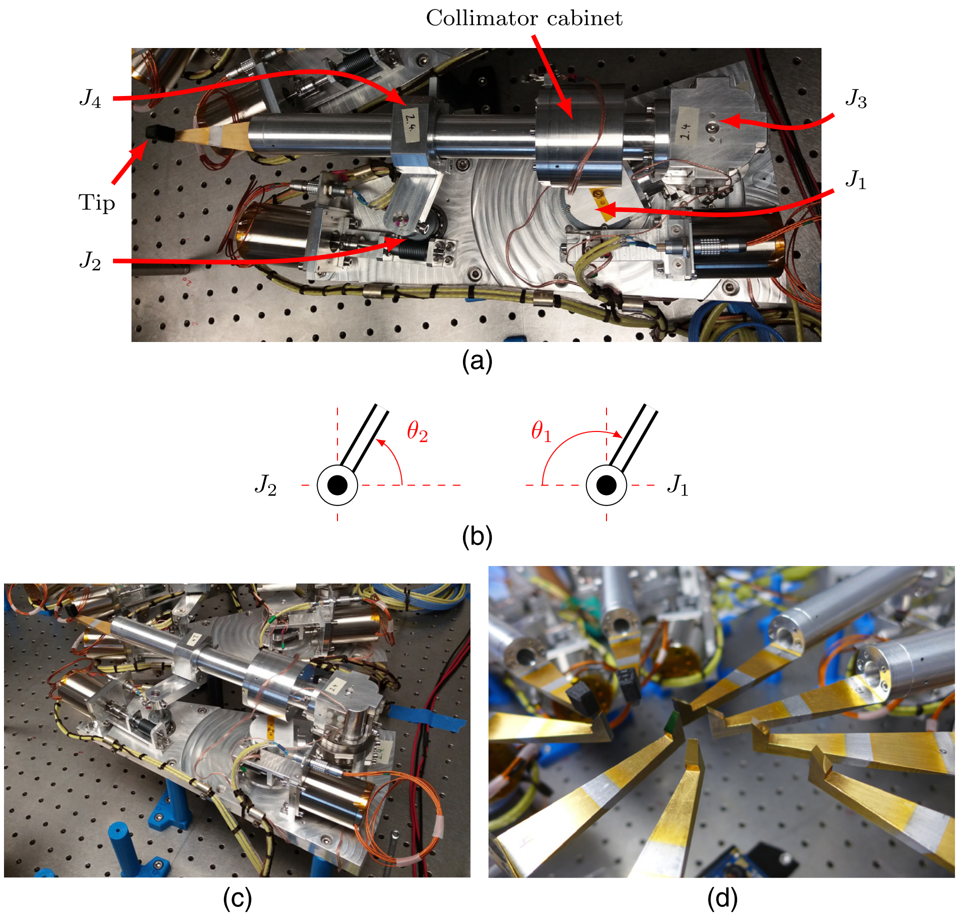


Fig. 3 The MIRADAS MXS probe arm. (a) An aerial view of a real arm with electrical wiring and motors. The four joints of the mechanism are labeled. J_1, J_2, and J_3 are revolute joints, whereas J_4 slides and rotates over the tube, where the collimator cabinet is present. A protection black cap is on the pickoff mirror to prevent scratches while the arm is not in operation. (b) The rotation angle θ_1 of joint J_1 increments counterclockwise, whereas θ_2 increases clockwise. (c) A probe from a lateral view. (d) Several arms. Here, the pickoff mirrors of each arm can be appreciated.

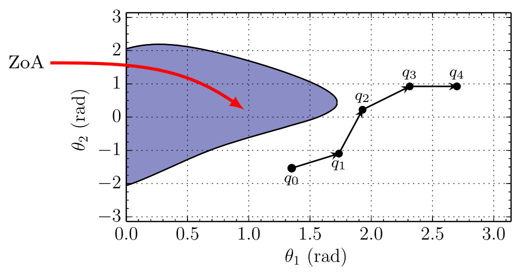


Fig. 4 The arm C-space comprises the set of all (θ_1, θ_2) combinations. However, there is a subset of them, the ZoA, that is not feasible due to constraints in the mechanical design of the arm. The trajectories are determined in this control space as sequences of points q_i, each of which with an associated time profile.

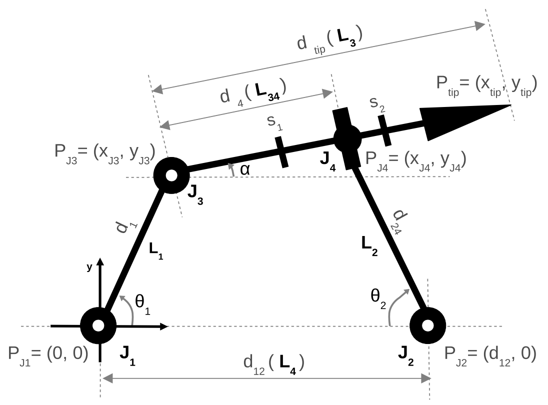


Fig. 5 The probe arm model consists of several constrained linkages. The arm frame is at joint J_1 and the position of joint J_4 must be always between stop position s_1 (collimator cabinet) and s_2 (end of long tube). The articular variables θ_1 and θ_2 denote the degrees-of-freedom of the mechanism.

$$x_{\text{tip}}(\theta_1, \theta_2) = d_{13} \cos \theta_1 + d_{3\text{tip}} \frac{1}{\sqrt{1 + u(\theta_1, \theta_2)^2}}, \tag{1}$$

$$y_{\text{tip}}(\theta_1, \theta_2) = d_{13} \sin \theta_1 + d_{3\text{tip}} \frac{u(\theta_1, \theta_2)}{\sqrt{1 + u(\theta_1, \theta_2)^2}}, \tag{2}$$

where u is

$$u(\theta_1, \theta_2) = \frac{-d_{13} \sin \theta_1 + d_{24} \sin \theta_2}{d_{12} - d_{13} \cos \theta_1 - d_{24} \cos \theta_2} \tag{3}$$

and the position $(x_{\text{tip}}, y_{\text{tip}})$ of the tip of the arm is expressed in terms of a local coordinate frame attached to the joint J_1. But, as commented earlier, not all pairs (θ_1, θ_2) are mechanically feasible. Hence, the solution is subject to the constraint $d_{s_1} < d_{34} < d_{s_2}$, where the value d_{34} for a given (θ_1, θ_2) pair can be obtained using the following expression:

$$d_{34}(\theta_1, \theta_2) = \sqrt{(d_{12} - d_{24} \cos \theta_2 - d_{13} \cos \theta_1)^2 + (d_{24} \sin \theta_2 - d_{13} \sin \theta_1)^2}. \tag{4}$$

This constraint creates the prohibited zone present in Fig. 4.

3.2 *Controlling the Motion*

The motion of the arm is controlled by the rotation of joints J_1 and J_2, each is actuated by a different stepper motor. Both steppers are wired to the same industrial dual-axis motor controller. A number of switches are present in the arm. Joint J_1 has two limit switches and a datum switch, whereas J_2, whose motion is not mechanically constrained, has only a datum switch. In addition, there is a limit switch attached to the collimator cabinet. With the help of these switches, a calibration procedure determines the real range of motion of the arm, including its associated zone of avoidance (ZoA). Motor steps are counted from their respective datum positions.

3.2.1 *Motor controller constraint*

As shown in Fig. 4, each trajectory defines a motion guided through a sequence of waypoints expressed in the arms C-space. Therefore, in order to execute trajectories of that kind, a motor controller should be capable of interpolating through those waypoints such that angular velocities and accelerations are smooth (differentiable). But, the commercial off-the-shelf controller does not have that feature. The only available way to connect any two consecutive points is through a linear segment, an assumption also found in motion planning specialized literature. Unfortunately, this approach delivers trajectories with discontinuities at the end of each segment as velocities cannot be switched instantaneously, forcing the arm to stop and restart its motion. Since these start–stop cycles result in extra arm joints wear, additional energy consumption and undesired increment in arm's travel time should be kept to a minimum.

4 Problem Description

A description of our problem follows. We refer to the m independent parameters that specify the position of an arm by the name of configuration. A configuration is denoted by q. The m-dimensional space containing all possible configurations is called the configuration-space (C-space). We consider a system composed of any given integer number n of articulated arms $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2 \dots \mathcal{A}_n\}$ moving in a common three-dimensional environment represented by $\mathcal{W}$. Each arm $\mathcal{A}_i$ patrols a reduced area of this environment and can move at a maximum velocity $v_{\max_i}$. Each arm $\mathcal{A}_i$ has its own C-space represented by $\mathcal{C}_i$. In addition, for each arm $\mathcal{A}_i$, there is an area in $\mathcal{C}_i$ prohibited due to mechanical constraints that we will denote by $\mathcal{C}_i^{\text{ZoA}}$. The subset of $\mathcal{C}_i$ containing all the mechanically feasible configurations is represented by $\mathcal{C}_i^{\text{feas}}$. We use the notation $\mathcal{A}_i(q)$ to refer to the volume in the common environment occupied by the arm $\mathcal{A}_i$ at configuration $q \in \mathcal{C}_i^{\text{feas}}$. Each robot has been given a starting and final configuration, q_{s_i} and q_{f_i}, respectively, both belonging to $\mathcal{C}_i^{\text{feas}}$. For convenience, we reduce, for each arm $\mathcal{A}_i$, its available $\mathcal{C}_i^{\text{feas}}$ into a smaller set known as a roadmap, represented by $\mathcal{R}_i$, which also contains q_{s_i} and q_{f_i}; see Sec. 6.1 for an in-depth discussion about how a roadmap is constructed.

We define a path from a given q_{s_i} to a q_{f_i} as being a continuous sequence of configurations entirely belonging to $\mathcal{R}_i$ and a trajectory as a path parametrized by time. A trajectory for $\mathcal{A}_i$ is formally expressed as $\pi_i : t \in [0, T_i] \to \pi_i(t) \in \mathcal{R}_i$, such that $\pi_i(0) = q_{s_i}$ and $\pi_i(T_i) = q_{f_i}$. The variable t denotes time and T_i the travel time, the instant when $\mathcal{A}_i$ reaches its destination and remains there. For convenience, these trajectories will be parametrically expressed in terms of a time sample $k \in \mathbb{N} \cup \{0\}$, where $\mathbb{N}$ is the set of natural numbers. The time step between two consecutive samples is denoted by Δt. Then, the continuous function $\pi_i(t)$ becomes the discrete function $\pi_i(k) = \{q_{i_0}, q_{i_1} \dots q_{i_n}\}$, where $q_{i_j} \in \mathcal{R}_i$ is the configuration of $\mathcal{A}_i$ when $k = j$. This way, $\pi_i(k)$ is defined by a number of waypoints for which $\mathcal{A}_i$ has to pass through at a particular time $k * \Delta t$. Two trajectories π_i and π_j are said to be safe if the arms following them do not collide and neither of them moves faster than its corresponding maximum velocity.

By configuration time, we understand the time required for all arms to move from their initial positions to their goals. This time, frequently known as makespan, is defined as the travel time of the last arm reaching its destination. For a given trajectory set $\mathcal{T} = \{\pi_1, \dots, \pi_n\}$, it can be computed as

$$T_{\mathcal{T}} = \max_{\forall\, i \in \mathcal{A}} k_i^{tr}$$

where k_i^{tr} denotes the discrete time step in which the arm $\mathcal{A}_i$ reaches its destination.

Finally, we assume that initial and final positions for each arm come from two different feasible assignment plans delivered by the target allocator. We also assume that target allocator has effectively performed its task; see Sec. 2.1 for details. Therefore, there is no collision between any two initial and any two final positions as well as the minimum distance between the targets is met.

Problem 1 (motion planning along roadmaps): Taking into account all previous definitions and assumptions, the problem we want to solve is the following: given a set of n arms $\mathcal{A}_i$, where n can be any integer number, each controlled by a device showing the start–stop behavior discussed in Sec. 3.2.1 and each with a roadmap $\mathcal{R}_i$ and a task $\langle q_{s_i}, q_{f_i} \rangle$ to be accomplished, find a set $\mathcal{T} = \{\pi_1, \dots, \pi_n\}$ such that fulfill:

1. No two trajectories, π_i and π_j, are unsafe $\to$ Requirement.
2. The number of start–stop cycles in each π_i is not greater than a given $nSSC$ $\to$ Requirement.
3. The configuration time is equal to or less than a given value T_p, set to 120 s $\to$ goal.

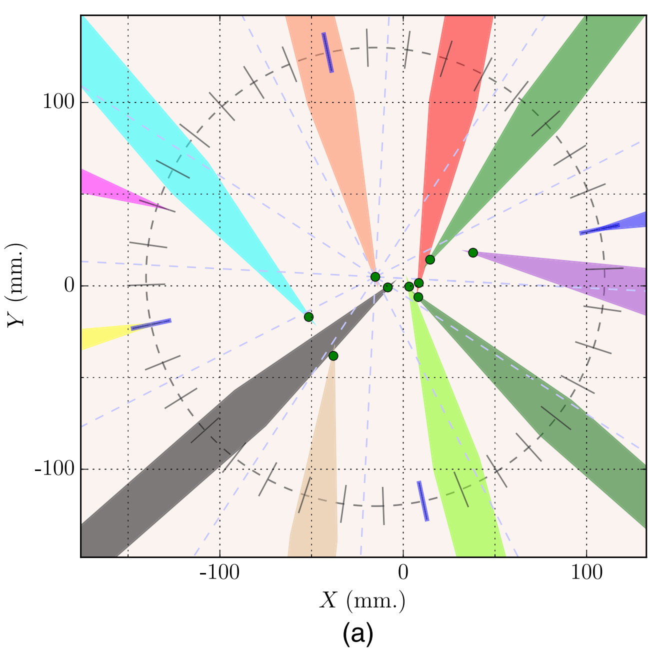
Points (2) and (3) are not related to science but to the efficiency of the instrument. While point (3) originally was a strict requirement for the MXS motion planner,[10] we are now treating this as a goal.

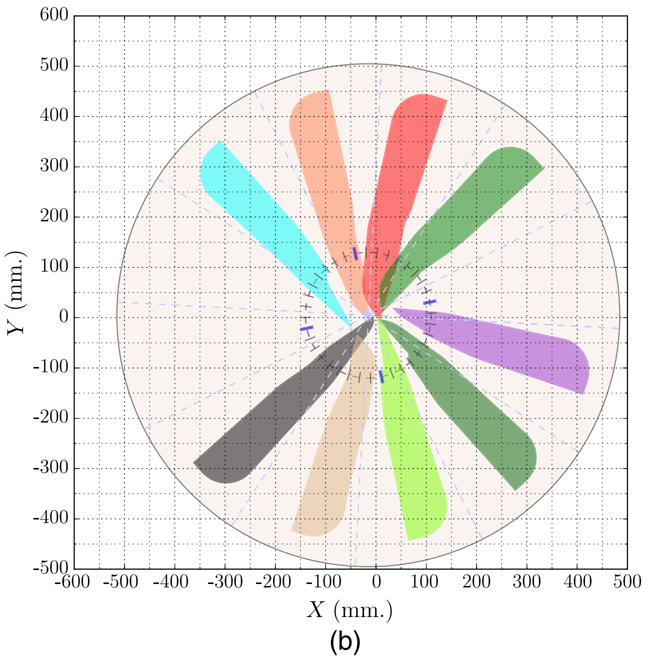
4.1 *Sensing the Real World*

To provide a high degree of flexibility and portability, the motion planning algorithm will only retrieve information about the n arms it can work with through a number of Boolean functions. We can distinguish here functions of two kinds:

1. Those checking if a given configuration $q \in \mathcal{C}_i$ is contained in the $\mathcal{C}_i^{\mathrm{ZoA}}$ of a given arm $\mathcal{A}_i$: `IsInZoA(q)`.
2. Those reporting if a given arm $\mathcal{A}_i$ configured at $q \in \mathcal{C}_i^{\mathrm{feas}}$ collides with the others at time t: `Collision(q,t)`.

The algorithm will need as many functions of each type as arms have to be planned. That is, it will require n functions `IsInZoA(q)`, each providing particular information about a distinct arm, as well as n functions `Collision(q,t)`, one for each arm. Each function `IsInZoA(q)` considers the





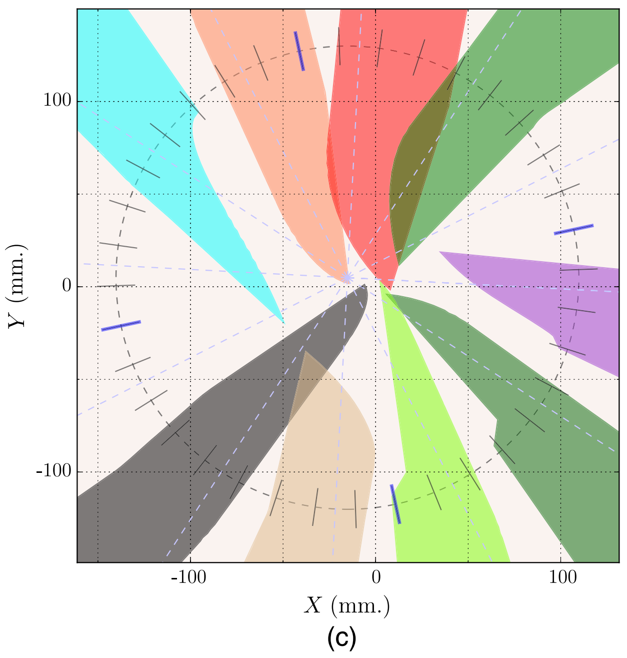


Fig. 6 The space-time set $\mathcal{O}^{\Delta}$ containing nine trajectories projected in the 2-D Cartesian space. (a) A close look of the MXS bench with the arms at their destinations. Nine out of twelve probes have been assigned to targets (dots). The circle represents the MIRADAS focal plane. (b) The nine different zones in the MXS bench show the areas swept by the assigned arms when moving from their initial positions to their final ones. (c) Detail of the swept areas. Although a few of these areas intersect, the arms will not collide if a proper time profile is found.

real ZoA of each arm as computed by the calibration procedure mentioned in Sec. 3.2. All functions `Collision(q,t)` have to take into account a safety tolerance, as explained in Sec. 2.1. For the motion planner, these $n \times 2$ functions are black boxes hiding the particular kinematic characteristics of each arm in the system. Every time the planner requires information from a particular arm, it will call the appropriate function.

5 Prioritized Planning

The trajectories determination problem defined in Sec. 4 is a multirobot motion planning problem (MMPP). The different approaches to MMPP can be broadly classified into coupled and decoupled. The former is an extension of the single-robot motion planning problem, where the whole system is modeled as a composite robot combining the states of all individual robots. Then, the paths for all of them are jointly determined with the help of a single-robot planner. This combined state space, containing all possible paths, guarantees to find a solution if it exists (completeness). Additionally, optimality can also be obtained if the combined space is used together with a graph search algorithm like A^*[13] or one of its more recent variants.[14] However, this full space is exponential in the number of robots. Thus, coupled approaches generally become computationally intractable with systems of many robots. By contrast, decoupled planners tend to decrease the dimensionality of the problem by planning the motions of each robot independently and then coordinating them to avoid collisions. They are faster, but, unfortunately, they are also incomplete.

Prioritized planning is a practical and often effective decoupled technique. This strategy, introduced by Erdmann and Lozano-Pérez,[15] assigns a priority to each robot and then determines their motions sequentially. The highest priority robot is planned first and after, in decreasing order, the lower priority ones compute their trajectories considering the previously planned robots as moving obstacles. Such a greedy technique is incomplete, and its performance is sensitive to the ordering of the agents. Some works study different priority schemes.[16,17] Silver goes one step further by exploring dynamic priorities so that every agent has the highest priority for a short period.[18] But often in some environments, simple heuristics perform well enough.[19]

In Sec. 5.2, we present high-level pseudocode for the prioritized planning approach. However, before discussing the algorithm, few concepts and notations need to be introduced.

5.1 *Notation*

We will work with the concept of space–time domain, which is a four-dimension space combining the 3-D environment, where all arms move ($\mathcal{W}$), with time. It is denoted by $\mathcal{Y}$ and formally defined as $\mathcal{Y} = \mathcal{W} \times t$, where $t \in [0, \infty)$. When an arm follows a given trajectory, it can be thought of as a dynamic object occupying a particular region in $\mathcal{Y}$. In addition, the obstacle-time subset, denoted by $\mathcal{O}^{\Delta}$, will contain the space–time regions occupied by all previously planner arms; see Fig. 6. Finally, we will employ the notation $\mathcal{A}_i^{\Delta}(\pi_i)$ for the transformation determining the space–time region swept by the arm $\mathcal{A}_i$ when moving along the trajectory π_i.

5.2 *Algorithm*

Two elements are required to build a prioritized solution: (i) a strategy to prioritize the arms and (ii) a method to compute the motions of a single arm in a known dynamic environment. First,

Algorithm 1 Arm-prioritized planning.

Input: $\mathcal{A}$: A set of *n* arms ordered by priority, $\mathcal{R}$: a set of *n* roadmaps

Output: $\mathcal{T}$: the set of trajectories for $\mathcal{A}$ if it exists. Otherwise failure ($\varnothing$)

```
1: function PRIORITIZED-PLANNING ($\mathcal{A}, \mathcal{R}$)
2:   $\mathcal{T} \leftarrow \varnothing$
3:   $\mathcal{O}^{\Delta} \leftarrow \varnothing$    ▷ No trajectories planned yet, so empty obstacle-time set
4:   for i ← 1 to n do
5:     $\pi_i \leftarrow$ ARM-MOTION-PLAN ($\mathcal{A}_i$, $\mathcal{R}_i$, $\mathcal{O}^{\Delta}$)    ▷ Plan trajectory with current obstacle-time set
6:     If $\pi_i = \varnothing$ then    ▷ Has a trajectory been found for the current arm?
7:       return $\varnothing$    ▷ Failure
8:     end if
9:     $\mathcal{T} \leftarrow \mathcal{T} \cup \{\pi_i\}$    ▷ Trajectory found, so update the set
10:    $\mathcal{O}^{\Delta} \leftarrow \mathcal{O}^{\Delta} \cup \mathcal{A}_i^{\Delta}(\pi_i)$    ▷ Update obstacle-time set with the new trajectory $\pi_i$
11:  end for
12:  return $\mathcal{T}$    ▷ Success: trajectories found for all arms!
13: end function
```

a priority scheme for the arms is determined. In practice, any method for assigning priorities can be employed. However, initially, we will use one appropriate for the purpose of minimizing the system makespan; see Sec. 7. Once the arms are arranged in priority descending order, it can be called the function `Prioritized-Planning`, which iteratively plans trajectories for each of them according to their priority; see Algorithm 1. The trajectory of each arm is determined by `Arm-Motion-Plan` (line 5), a single arm roadmap-based motion planner described in Sec. 6. This method tries to find a trajectory in the arm's roadmap $\mathcal{R}_i$ such that avoids the space–time regions occupied by the previously planned arms (obstacle-time set $\mathcal{O}^{\Delta}$). If `Arm-Motion-Plan` fails to find a safe trajectory for the current arm, then `Prioritized-Planning` terminates with failure (line 7). Otherwise, it tries to determine a motion plan for the next arm.

6 Planning an Arm Trajectory

In this section, we present a roadmap-based motion planner for a single arm moving in a dynamic environment known beforehand. Algorithms relying on sampled-based roadmaps, such as probabilistic roadmap[20] and rapidly exploring random tree,[21] and their multiple variants have been successfully demonstrated on different robotic platforms. They construct roadmaps by sampling C-space and checking their validity with the help of a function sensing the environment. By limiting the motions of the robots to the paths present in the roadmap, the search space that has to be explored to find a feasible trajectory is conveniently reduced. Once the roadmap is built, a standard graph-search technique can be employed to find the best path connecting two points of the roadmap according to a given criteria.

These algorithms are said to be probabilistically complete as its probability of failure exponentially tends to be zero when the number of samples in the roadmap approaches to infinity. Furthermore, the quality of the trajectories computed highly depends on the quality of the roadmaps.[22] If roadmaps accurately model the real connectivity of the mechanism C-space, this kind of planner offers a good compromise between being practically usable in complex scenarios and completeness.

6.1 *Roadmap*

As its name indicates, a roadmap is a smaller set of the original arm feasible C-space containing a series of alternative paths connecting a starting and a goal configuration. Every path comes determined by two elements: its vertices and its edges. The former of these represents waypoints in the path. They can be the initial, the final, or any intermediate point in the path. The edges model the segments of the path that connect every two consecutive waypoints. In principle, these segments can have any shape, which will depend on the interpolation method utilized to connect the two vertices involved. Here, any polynomial function producing smooth transitions or an ad-hoc heuristic search[23,24] could be effectively applied. However, for their simplicity, straight-lines have frequently been used in theoretical works. In our case, due to the problem constraint presented in Sec. 3.2.1, all edges will also be straight-lines.

A roadmap is expressed in terms of a graph $\mathcal{R} = (V, E)$ formed by the set V containing the vertices of all paths and the set E containing the edges of all paths. By sampling the arm C-space with the appropriate `IsInZoA(q)` function, introduced in Sec. 4.1, paths are incrementally constructed following a process involving three steps. The two first of them do not take into account the initial and final configuration of the arm. In the first step, the set of all vertices in the roadmap is determined. There exists in literature a spectrum of approaches, ranging from pseudorandom to grids, about how to compute this set.[25] We use a grid similar to the one in Fig. 7(a) and check each configuration belonging to a junction for feasibility. Those configurations in the arm's ZoA are automatically discarded while the rest are included in the set V.

Then, in the second step, we determine which pairs of vertices of V will be connected, resulting this way in a different number of edges. A predefined number of pairs (q_i, q_j) from V are stochastically chosen. If the straight-line connecting a given q_i and a given q_j do not traverse ZoA, the edge $q_i \rightarrow q_j$ is added to the roadmap edge set E. When selecting these pairs, we impose the constraint that a given vertex $q_i \in V$ cannot be present in more than one resulting edge.

Finally, in the third step, the initial q_{ini} and final q_{end} configurations of the arm are considered to form several paths between them involving a different number of edges. First, q_{ini} and q_{end} are properly connected to every edge in the previously computed set E, delivering paths with three edges $(q_{\text{ini}} \rightarrow q_i \rightarrow q_j \rightarrow q_{\text{end}})$. If the corresponding edges $q_{\text{ini}} \rightarrow q_i$ and $q_j \rightarrow q_{\text{end}}$ do not traverse ZoA, then the path is included in the roadmap. Furthermore, the q_{ini} and q_{end} points are connected to every vertex q_i of the set V, resulting multiple paths with two edges $(q_{\text{ini}} \rightarrow q_i \rightarrow q_{\text{end}})$. If the edges of these paths are mechanically feasible, the paths are added to the roadmap.

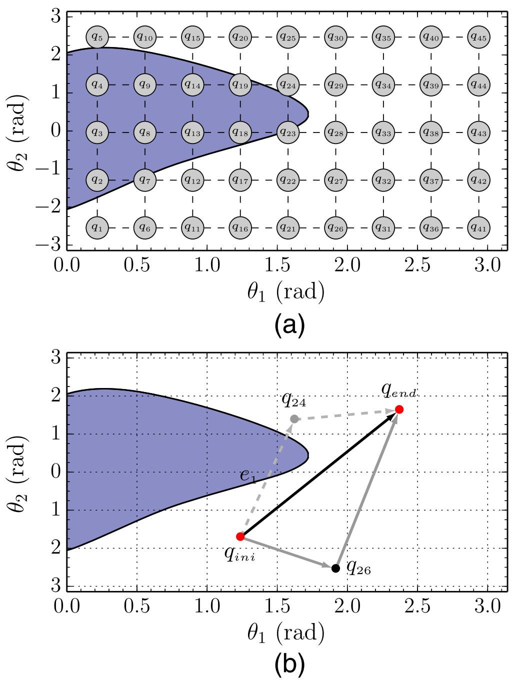


Fig. 7 Sampling C-space and roadmap paths. (a) The arm C-space is sampled using a grid. Here, as an example, we show a particular 9×5 grid. All those configurations q_i not in the ZoA will be included as vertices in the arm roadmap. Additionally, a random number of pair of vertices (q_i, q_j) are connected and included in the roadmap's edge set. (b) Three paths connecting the initial (q_{ini}) and final (q_{end}) position of an arm in C-space. The feasibility of each edge is checked. If any passes through ZoA, then the path containing it is automatically discarded. In this case, the path $q_{\text{ini}} \rightarrow q_{24} \rightarrow q_{\text{end}}$ is discarded due to the infeasibility of e_1 $(q_{\text{ini}} \rightarrow q_{24})$.

Finally, a direct path linking together q_{ini} and q_{end} is also included if it is feasible. In Fig. 7(b), three different paths are represented in C-space.

6.2 *Planner*

Our planner uses a two-level approach to compute a collision-free trajectory. In the first of them, the most promising unexplored path connecting the origin and destination of the arm is selected from the arm's sampled-based roadmap. Then, on the local level, the motions of the arm along that path are appropriately coordinated with those of the previously planned arms. As we have seen, a path might be composed of several edges, each of which will be adequately discretized and systematically explored. Finally, our solution incrementally generates the search space as it directly depends on the previous motions performed by the arm.

The pseudocode of the proposed motion planner is given in Algorithm 2. It iterates over the paths in the arm's roadmap (line 2), from the most promising in terms of estimated travel time to the least one. If two or more paths present the same estimation, one of them is arbitrarily selected. Then, the edges in a path are sequentially explored (line 6) by the method `Find-Edge-Traj`, as discussed in Sec. 6.4. If it founds a safe trajectory for the current edge in the path, then the next one is visited. Otherwise (line 8), the remaining edges are skipped and the next path in the roadmap is checked. The search ends when collision-free trajectories have been found for every edge in a path or the entire roadmap has been unsuccessfully visited.

Algorithm 2 Finding a trajectory for one arm.

Input: $\mathcal{A}_i$: a given probe arm, $\mathcal{R}_i$: arm's roadmap, $\mathcal{O}^\Delta$: *obstacle-time* set

Output: π_i: a trajectory for $\mathcal{A}_i$ if it exists. Otherwise, failure ($\varnothing$)

```
1:  function ARM-MOTION-PLAN ($\mathcal{A}_i$, $\mathcal{R}_i$, $\mathcal{O}^\Delta$)
2:    for all p ∈ $\mathcal{R}_i$ do        ▷ Explore paths in Roadmap: most promising first
3:      t←0        ▷ Travel time
4:      $\pi_i$←∅
5:      for all e ∈ p do        ▷ Explore all edges in a path
6:        $\langle \pi_e, t_e \rangle$← FIND-EDGE-TRAJ ($e, \mathcal{A}_i, \mathcal{O}^\Delta, t$)
7:        if $\pi_e = \varnothing$ then        ▷ Check if traj. found for the current edge
8:          Exit loop        ▷ Skip remaining edges in cur. path
9:        end if
10:       $\pi_i$←$\pi_i \cup \{\pi_e\}$        ▷ Update traj. with local traj. along the edge
11:       t ← t + $t_e$        ▷ Update travel time with the time spent in the edge
12:     end for
13:     if trajectories found for all edges in a path then
14:       return $\pi_i$        ▷ Trajectory found!
15:     end if
16:   end for
17:   return ∅        ▷ Failure: no trajectory found for this arm
18: end function
```

6.3 *Estimating Path Travel Time*

The travel time of a given roadmap path is computed as the sum of the estimated travel times of each of its parts. A lower-bound estimate for the travel time of a roadmap edge e_{ij} connecting arm configuration $q_i = (\theta_1^i, \theta_2^i)$ and $q_j = (\theta_1^j, \theta_2^j)$ can be determined as follows:

$$t_{e_{ij}} = \max_{\forall\ k \in [1,2]} \frac{|\theta_k^i - \theta_k^j|}{v_{\max}^k}, \tag{5}$$

where θ_k is the k component (or degree-of-freedom) of an arm configuration and $v_{\max}^k$ is its maximum velocity.

6.4 *Exploring an Edge of the Roadmap*

In this section, we discuss how a trajectory along a given edge of the roadmap is explored to determine if it can be safely followed by the arm. The method we employ here is partially inspired by a grid-based planner for robotics application.[19] There, an edge of a roadmap is discretized into a mesh of states and connections representing valid transitions between these states. This convenient representation enables addressing motion planning as a graph search problem. Our approach differs from that work in that we do not invariably use a three-connected grid. The neighbors of each state are dynamically computed considering the number of start–stop cycles accumulated during the previous motions. Furthermore, our approach is also able to limit the maximum number of start–stop cycles per trajectory. Finally, we also use different granularities for the motion action time step and for collision checking.

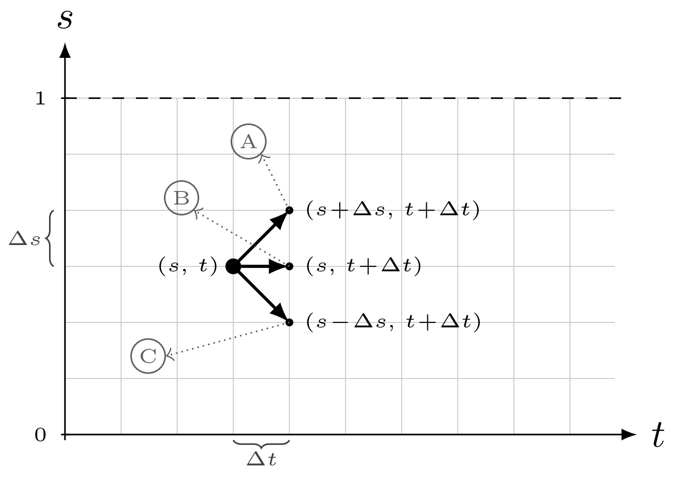
6.4.1 *State-time space*

We use the notation $G = (X, C)$ to denote the graph G dynamically built to coordinate the motions of an arm along a given roadmap edge. The set X represents the nodes of the graph, and C is the set of connections between each pair of nodes. Each node $z = (x, t)$ is formed by a compound element x defining the state of the arm and a scalar t specifying a notion of time. The domain containing all feasible pairs (x, t) is known as state-time space. Since its inception, this representation[15,26] has been broadly used for motion planning in dynamic environments.

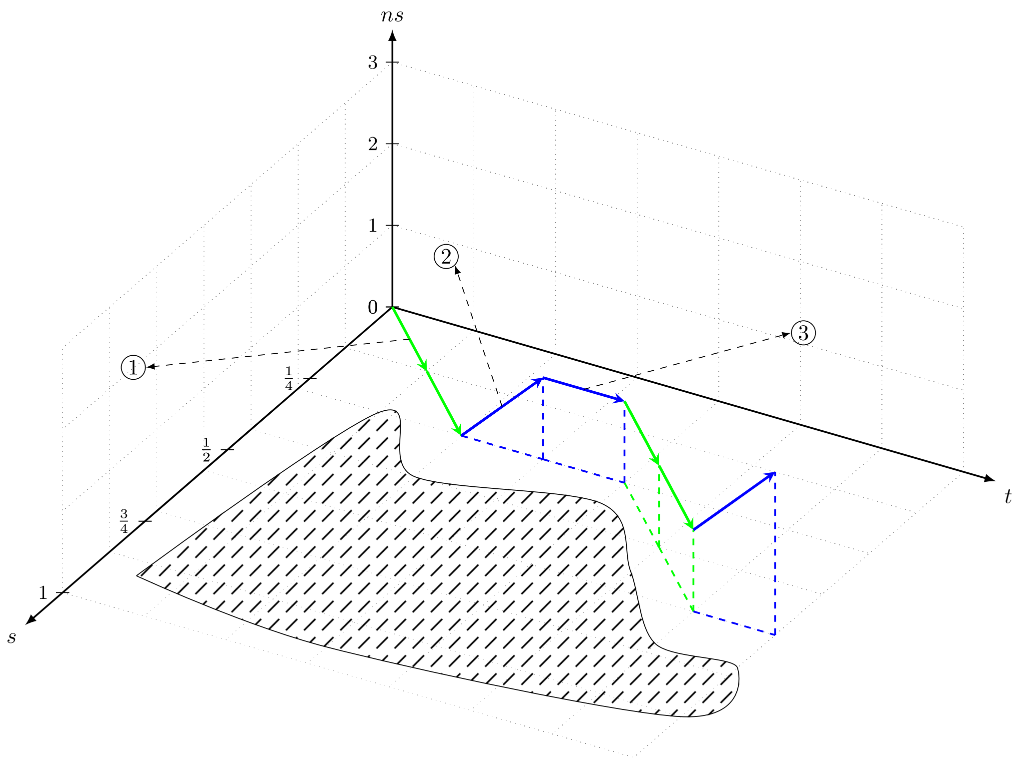
In our case, all the possible configurations an arm can adopt when moving along a roadmap edge have been conveniently reduced to a variable s, which ranges from 0 to 1. This variable specifies the distance traveled along the edge, where $s = 0$ and $s = 1$, respectively, represent the configurations of the source and destination vertices of that edge. Then, the state of an arm is defined as a three-tuple (s, ns, m), where ns denotes the number of start–stop cycles the arm has performed and, finally, m is the motion action used to reach this state. This way the arm being planned is represented by a point in four-dimensional state-time space and the previously planned arms are treated in each particular instant of time as static objects.

Discretization: To perform a discrete-time analysis, we will follow the approach of LaValle and Hutchison.[27] We select a global time step Δt. As a result, an edge e_{ij} of length d can be approximated by a finite sequence containing n points. Each of these points corresponds to the arm's configuration at instant $k\Delta t$, where $k \in \{0,1\ldots n-1\}$. We select then $v_{e_{ij}}$ for e_{ij} as the greatest value smaller than $v_{\max}$ so that the edge is divided into an integer number of segments. Consequently, the normalized length of each of those segments is $\Delta s = |v_{e_{ij}}|\Delta t/d$, being d the length of the edge e_{ij}.

Motion actions and graph connections: It is assumed that the motor controller of an arm can process a command each Δt s. The arm, therefore, is only allowed to change velocity at each instant $k\Delta t$ and the values to choose from are constrained to $-v_{e_{ij}}$, 0, and $v_{e_{ij}}$. That is, the arm can decide to move Δs backward along the edge, to remain motionless or to move Δs forward. These motion actions determine the neighbors a graph node can reach. In practice, connections between states must always move forward in time. Thus, if considering only the s component of the arm state, the three potential neighbors of a given state-time node (s, t) are $(s - \Delta s, t + \Delta t)$, $(s, t + \Delta t)$, and $(s + \Delta s, t + \Delta t)$. As shown in Fig. 8(a), this defines a grid of potential graph connections to consider when trying to find a safe trajectory along the edge. However, not all these connections might be valid. When the maximum allowed number of start–stop cycles has been reached, the arm has to be always forced to move in the same direction. That is the main reason why the nodes of the graph store also information about the motion action m. This variable can only take three possible



(a)



(b)

Fig. 8 The state-time search space of a roadmap edge. (a) The grid shows the potential neighbors that can be reached from a given state-time node (s, t), where $s \in [0,1]$ denotes the roadmap edge distance and t denotes a scalar time step (Δt). The neighbors labeled as A, B, and C, respectively, represent a forward motion along the roadmap edge, no motion, and a backward motion. Each of those atomic actions has a duration of one time step. (b) The search space including a third dimension, the start–stop cycles counter (*ns*). If the arm is in motion, this counter is incremented every time a motion action different from the current one happens. The arrows show the different atomic motion actions to be executed by the arm each Δt seconds, and the dashed area defines an obstacle-time object to be avoided. The label 1 shows a forward motion. The label 2 represents a stop after a forward motion, therefore, the *ns* counter is incremented. The label 3 specifies a motionless (or stop) action. The sequence of motion actions shown in the figure are: forward (2 time steps), stop (2 time steps), forward (2 time steps) and stop (1 time step).

values $\{-1,0,1\}$, respectively, representing backward motion, no motion, or forward motion.

Although it will be discussed in depth in Sec. 6.4.2, we would like to introduce here that, in fact, the planning of motions along a roadmap edge can be seen as a search in a more general 3-D space. The additional axis of such search-space [see Fig. 8(b)] is one specifying the number of start–stop cycles (*ns*) accumulated. This state variable *ns* is of great significance since it impacts on the geometry of the final path found by the algorithm. Its maximum allowed value can adequately control the smoothness of the solution, which, as stated before, is a fundamental requirement of our design.

6.4.2 *Searching a trajectory in a roadmap edge*

As it has been stated in Sec. 6.4.1, the problem of finding a safe trajectory along a roadmap edge can be formulated as a graph search problem. The state-time grid has to be systematically explored to find a collision-free path connecting the (s, t) points $(0, t_s)$ and $(1, t_f)$, where 0 and 1, respectively, denote the source and destination vertices of the edge and t_s and t_f $(t_s < t_f)$, the initial and arrival time intervals. Since we are interested in obtaining a system makespan lower than a given value, we will try to achieve it by individually minimizing the travel time of each arm. In practice, the A^* algorithm[13] is the de facto choice when an optimal path connecting two nodes of a graph needs to be computed. However, in dynamic environments, a depth-first graph search (DFS) is guaranteed to be faster than A^* in computing trajectories reaching its destination as soon as possible.[19]

Unfortunately, arriving too early might be inconvenient, especially if the scenario presents goal conflicts. Conflicts of this kind occur when the path of an arm passes by the goal location of another one. No matter the prioritization, if a lower priority arm arrives at its goal earlier than others having higher priorities, then further actions need to be taken to ensure safety. To illustrate that, let us consider an arm $\mathcal{A}_i$ with priority p_i and another arm $\mathcal{A}_j$ with priority p_j, such that $p_i > p_j$. The collision-free trajectory for $\mathcal{A}_i$ and $\mathcal{A}_j$ are, respectively, denoted by π_i and π_j, being T_i and T_j the corresponding travel times. Then, if $T_i > T_j$, π_i and π_j are, by construction of Algorithm 1, safe during the time interval $[0, T_j]$, but, they might not be during $(T_j, T_i]$. Thus, the feasibility of π_j in the later interval must be explicitly checked.

Algorithm: The method `Find-Edge-Traj` computes the motion actions safely moving the arm along a roadmap edge. The algorithm manages status information and also backpointers to preceding nodes for each state-time node z generated during the search. The former, labeled as status ∈ {UNEXPLORED, SAFE, UNSAFE}, prevents revisiting unpromising nodes, whereas the later, labeled as parent, is required to retrieve the sequence of motion actions forming a successful trajectory. Its pseudocode is given in Algorithm 3.

`Find-Edge-Traj` first stores (line 2) the source vertex of the edge (z_s) in the sequence of state-time nodes to be explored $(\mathcal{N})$. Then, the algorithm loops over this sequence (line 3). The state-time nodes are consumed (line 4) and visited by the helper method Explore-Node (line 5) in LIFO order, which guarantees a DFS. The search finishes when one collision-free node being at the destination roadmap vertex $(s_i = 1)$ is found (line 6) or all state-time nodes have been unsuccessfully explored.

Algorithm 4 shows the pseudocode for method `Explore-Node`, which employs the appropriate collision checking routine `Collision`, as discussed in Sec. 4.1.

Each time it first explores a node z (z.status = UNEXPLORED), this node is checked for safety (line 5) against the environment (the *obstacle-time* set). If it is collision-free, then its neighbors are computed (method `Neighbors` in line 8) only if $t < T_f$. This upper bound of t is introduced to abort the search if a trajectory has not been found after a given number of time steps.

It is assumed that calling `Collision` every Δt, the moments of time each arm is allowed to change its velocity,

Algorithm 3 Searching a safe trajectory along a roadmap edge.

Input:

$z_s = (s_s, ns_s, m_s, t_s)$: a *state-time* node

nStop: max. *start-stop* cycles allowed

T_f: max. *time* to explore

Output: true if an edge trajectory exists. It can be retrieved using the data stored in the *parent* field of each node z.

```
1:  function FIND-EDGE-TRAJ ($z_s$, nStop, $T_f$)
2:    $\mathcal{N} \leftarrow \langle z_s \rangle$        ▷ Sequence of state-time nodes to explore
3:    while $\mathcal{N} \neq \emptyset$ do        ▷ Are nodes pending?
4:      $z_i \leftarrow \mathcal{N}[-1]$        ▷ Get the last elem. in sequence $\mathcal{N}$
5:      $\mathcal{N}_i \leftarrow$ EXPLORE-NODE ($z_i$, nStop, $T_f$)
6:      if $s_i = 1$ and $z_i$.status = SAFE then        ▷ If destination reached and no collision there
7:        return true        ▷ Trajectory found. Use backpointers to retrieve traj.
8:      end if
9:      $\mathcal{N} \leftarrow \mathcal{N} - \langle z_i \rangle$        ▷ Remove the current node from the sequence
10:     $\mathcal{N} \leftarrow \mathcal{N} + \mathcal{N}_i$        ▷ Add the neighbors at the end of the remaining nodes
11:   end while
12:   return false
13: end function
```

is enough. However, if the atomic arm motions have long duration, there is a chance that the arm being planned could jump over thin dynamic obstacles. In those cases that a finer resolution is required, a submultiple Δt_{col} of Δt should be used. Then, `Collision` would need to be redefined so that it accordingly interpolates the path connecting the current state-time state and its parent and checks its feasibility every Δt_{col}. Employing different resolutions can be very convenient as the local search can be accelerated (by reducing the branching factor) without compromising safety. Algorithms 5 and 6 determine the neighbors of the node being explored considering the start–stop cycles counter (ns). If the maximum value (nStop) has been reached, then the method `Neigh-maxNS` dynamically computes the neighbors. Otherwise, it is done by `Neigh-DF`. As can be appreciated in line 2 of Algorithm 6, the method `Neigh-maxNS` always generates a neighboring node containing the same motion action. In addition, if the arm is stopped $(m = 0)$, then the neighbor advancing toward the destination edge is also returned (line 6). However, when nStop was not reached, the ns variable is updated in `Neigh-DF` and all three neighbors are stored in the particular order shown in line 33 of Algorithm 6. This way of arranging the nodes

Algorithm 4 Exploring a state-time node.

```
Input:
  $z = (s, ns, m, t)$: a state-time node
  nStop: max. start-stop cycles allowed
  $T_f$: max. time to explore
Output:
  $\mathcal{N}$: a sequence with the neighbors of $z$ sorted from the least
     promising to the most promising.
  z.status: node status is updated to UNSAFE or SAFE
1:  function EXPLORE-NODE ($z$, nStop, $T_f$)
2:    $\mathcal{N} \leftarrow \emptyset$
3:    if z.status = UNEXPLORED then        ▷ Has the node been explored?
4:      z.status ← UNSAFE
5:      if not COLLISION $(s, t)$ then
6:        z.status ← SAFE     ▷ Node is collision-free, so update status
7:        if $t < T_f$ then     ▷ Get neighbors only if max. t has not been reached
8:          $\mathcal{N} \leftarrow$ NEIGHBORS ($z$, nStop)
9:        end if
10:     end if
11:   end if
12:   return $\mathcal{N}$
13: end function
```

Algorithm 5 Determining connections of a *state-time* node (part 1).

```
Input: $z = (s, ns, m, t)$: a state-time node, nStop: the max. start-stop
       cycles allowed
Output: $\mathcal{N}$: a sequence containing the neighbors of $z$
1:  function NEIGHBORS ($z$, nStop)
2:    if $ns = nStop$ then
3:      $\mathcal{N} \leftarrow$ NEIGH-MAXNS ($z$)     ▷ Returning neighbor in same direction
4:    else
5:      $\mathcal{N} \leftarrow$ NEIGH-DF ($z$)     ▷ Returning neighboring nodes in depth-first order
6:    end if
7:    return $\mathcal{N}$
8:  end function
```

Algorithm 6 Determining connections of a *state-time* node (part 2).

```
1:  function NEIGH-MAXNS($z$)
2:    $nz \leftarrow (s + m * \Delta s, ns, m, t + \Delta t)$     ▷ Neighbor keeps moving in the same direction
3:    nz.parent ← $z$     ▷ Store neighbor's parent state
4:    $\mathcal{N} \leftarrow \langle nz \rangle$
5:    if $m = 0$ then     ▷ If arm was stopped
6:      $nz \leftarrow (s + \Delta s, ns, m, t + \Delta t)$     ▷ Add also the forward neighbor
7:      nz.parent ← $z$
8:      $\mathcal{N} \leftarrow \mathcal{N} + \langle nz \rangle$
9:    end if
10: end function

11: function NEIGH-DF($z$)
12:   if $m = -1$ then     ▷ The node being explored has been reached moving backward ($m = -1$)
13:     $ns_{-1} \leftarrow ns$     ▷ Num. start-stop cycles performed to reach the backward neighbor
14:     $ns_0 \leftarrow ns + 1$     ▷ Num. start-stop cycles performed to reach the motionless neighbor
15:     $ns_{+1} \leftarrow ns + 1$     ▷ Num. start-stop cycles performed to reach the forward neighbor
16:   end if
17:   if $m_i = 0$ then     ▷ If arm was stopped, do not increment start-stop cycles
18:     $ns_{-1} \leftarrow ns$
19:     $ns_0 \leftarrow ns$
20:     $ns_{+1} \leftarrow ns$
21:   end if
22:   if $m_i = 1$ then
23:     $ns_{-1} \leftarrow ns + 1$
24:     $ns_0 \leftarrow ns + 1$
25:     $ns_{+1} \leftarrow ns$
26:   end if
27:   $nz_{-1} \leftarrow (s - \Delta s, ns_{-1}, -1, t + \Delta t)$     ▷ Computes backward neighbor
28:   $nz_{-1}$.parent ← $z$     ▷Store neighbor's parent state
```

```
29:     $nz_0 \leftarrow (s, ns_0, 0, t + \Delta t)$            ▷ Computes motion-less neighbor
30:     $nz_0$.parent $\leftarrow z$
31:     $nz_{+1} \leftarrow (s + \Delta s, ns_{+1}, 1, t + \Delta t)$      ▷ Computes forward neighbor
32:     $nz_{+1}$.parent $\leftarrow z$
33:     $\mathcal{N} \leftarrow \langle nz_{-1} \rangle + \langle nz_0 \rangle + \langle nz_{+1} \rangle$     ▷ Store at the end of the sequence $\mathcal{N}$ the forward neighbor
34:     return $\mathcal{N}$
35: end function
```

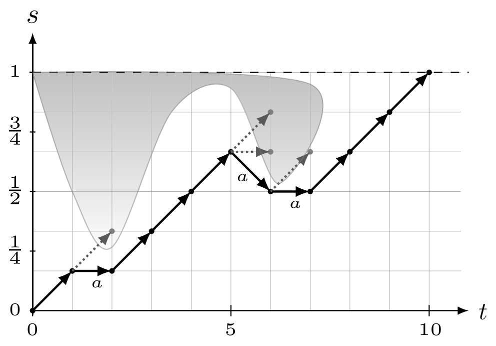


Fig. 9 DFS of a trajectory along a given roadmap edge considering only (s, t). The gray area represents a dynamic object expressed in state-time. The search starts at one edge end ($s = 0$) and completes when the other one ($s = 1$) is reached. The bold arrows show the resulting trajectory, whereas the dotted ones show those motion actions unsuccessfully explored. The neighbors are always visited in the following order: forward, no motion, and backward. Therefore, the DFS tries to get the arm to its final destination in a minimum number of Δt. If there is no constraint in *ns*, the trajectory found will closely surround the gray area, yielding, in general, a sequence of unsmooth motions. The actions labeled as *a* are those producing a start–stop cycle, consequently incrementing the *ns* counter.

along with the depth-search nature of Algorithm 3 ensures that the most promising motion action, the one approaching the arm toward the edge destination vertex [see label *A* in Fig. 8(a)], is always explored first.

Figure 9 illustrates how the method `Find-Edge-Traj` explores the state-time space. The effect of the state variable *ns* in the smoothness of the solution can be appreciated in Fig. 10, where searches along a roadmap edge are performed with different maximum values for that variable.

7 Prioritization

One of the key points when designing a prioritized planning algorithm is the determination of a priority scheme. We have adopted one that tries to minimize the makespan of the whole system by reducing the maneuvering of those arms whose optimal paths are longer. Let ϕ_i denote the optimal path of an arm A_i and T_{ϕ_i} its estimated travel time (see Sec. 6.3). This path directly connects the starting and final configurations, consequently, being the one in the roadmap sweeping the minimum area in $\mathcal{W}$. Thus, those arms having to traverse longer distances are planned first. That is, those arms presenting optimal paths with larger estimated travel times are assigned higher priorities.

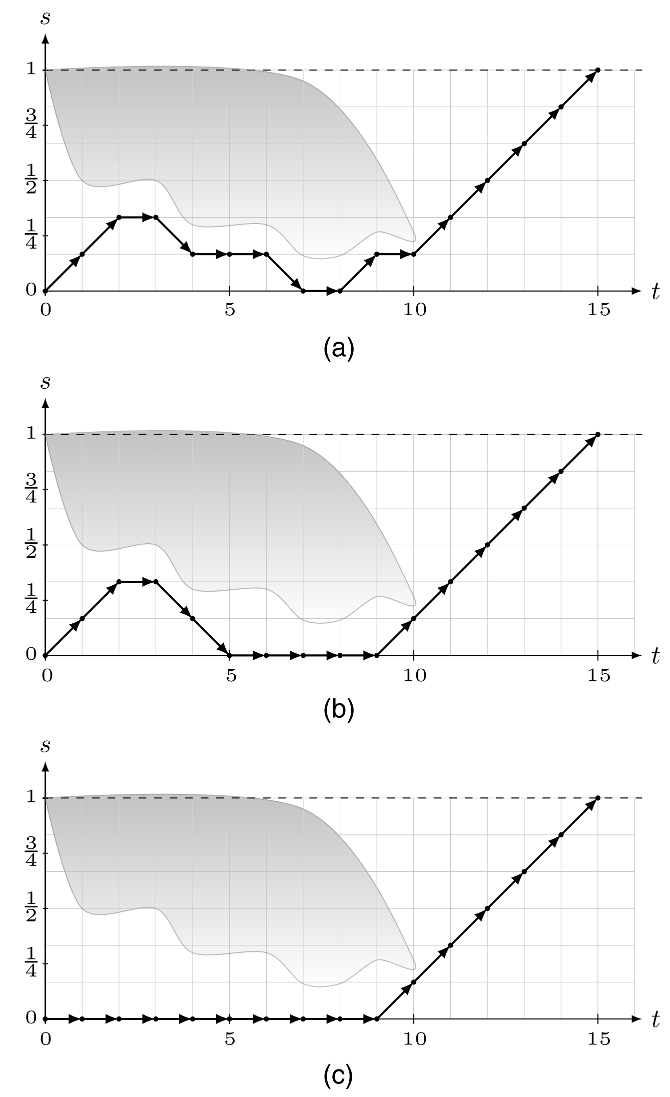


Fig. 10 How the variable *ns* impacts on the smoothness of the trajectory found along one edge. (a) The search is guided by the maximum number of start–stop cycles (*ns*) allowed. In this scenario, there is no any constraint in the maximum value the variable can take. Therefore, no smoothness control is considered when exploring the search space, yielding a final motion profile consisting of nine real motion actions (and four start–stop cycles). (b) Here, the maximum value of *ns* is constrained to 2, then the solution returned will perform two or less start–stop cycles. The real motion actions composing the trajectory are forward, stop, backward, stop, and forward. (c) Here, the maximum value of *ns* is constrained to 0. The resulting arm motion profile is formed by a motionless action during nine time steps followed by a forward action. This trajectory actually can be executed with a single real arm controller command.

8 Experimental Results

The proposed motion planning algorithm was implemented and tested in several scenarios with real science targets. At the end of this section, we take the reader through one typical scenario of the MXS system in detail. There, during their motions, the arms were exposed to multiple instances of all identified collision conflicts that can occur in MIRADAS.

In all cases, a particular roadmap was built for each arm in the system. Each dimension of the 2-D probe arm C-space was divided into equidistant segments, forming a 8×6 grid. Additional connections were set between random pairs of points in the grid, yielding more than 64 potential alternative routes. Then, the initial and final positions in C-space of each arm were connected to each of those points. Those paths with estimated travel time greater than 120 s or forcing the arm to pass through any infeasible configuration were automatically discarded. The maximum joint velocity used for each joint was 0.031416 rad/s, which are the maximum achievable by the real mechanism, as confirmed by laboratory tests. Finally, the maximum number of start–stop cycles allowed per trajectory was 1.

By design, the running time of our solution tightly depends on the collision checking time step (Δt_{col}) and on the motion action time step (Δt). We employed different resolutions for each of them. The value of Δt was chosen taking into account the duration of one of the longest direct motions an arm can execute. That motion takes around 100 s, as empirically determined with real hardware. The value utilized for Δt was 6 s, assuring a good trade-off between arm maneuverability and graph branching factor. Finally, the value of Δt_{col} used in the experiments was 100 ms. That figure was considered small enough in relation to the real velocities of the arms to detect collision between two different bodies correctly.

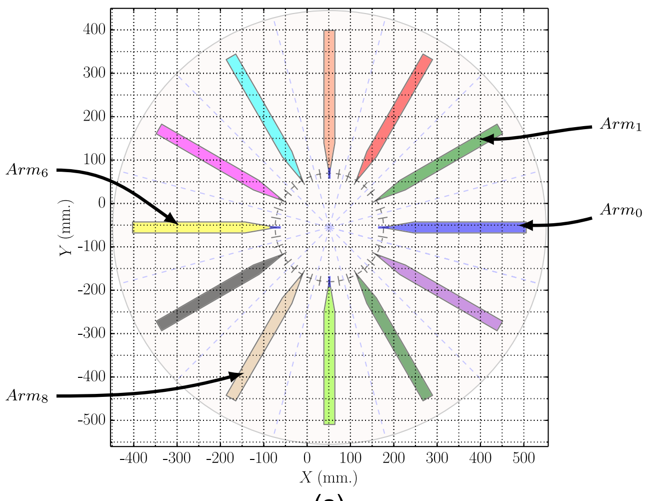
Finally, we use the target allocator to generate 200 different assignment plans containing real science objects. We compute trajectories for these plans in two different experiments. In the first of them, for each assignment plan, the arms start their motions from their corresponding park positions [see Fig. 11(a)] and end at their assigned targets. Before determining the trajectories for each of the assignment plans, the arms were numerically prioritized according to the estimated travel of their corresponding optimal paths. The higher that time, the greater the priority assigned to the arm. We find that in the 97% of the assignment plans, the algorithm successfully computes collision free-trajectories with a configuration time inferior to 120 s. The median configuration time is 102.55 s, ranging from a minimum of 96 s to a maximum of 114 s.

In the second experiment, all arms start their trajectories at their park locations and finish them at the targets of the first assignment plan. From there, the arms move to the second assignment plan and so on. As in the previous case, before determining the corresponding trajectories, arms are prioritized according to their estimated travel times. In this case, 93% of all assignments were successfully planed with a median configuration time of 90.6 s and a maximum of 108 s. The median value here is smaller than in the previous case because arms now start the next set of observations from a closer position. However, the success rate is inferior since arms are closer and consequently the probability of collision is greater.

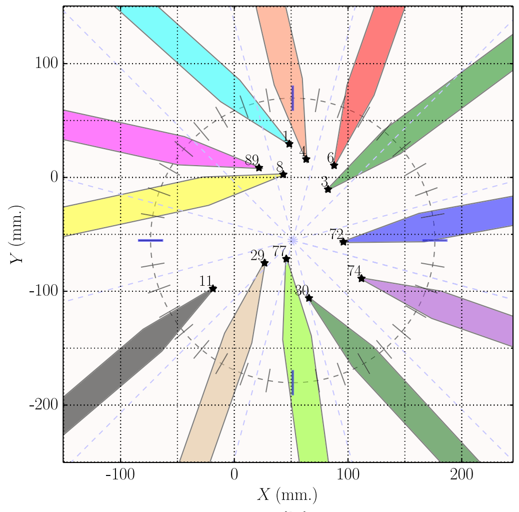
Please note that in all the previous tests, we have only run the motion planning algorithm once, as the purpose was to show the algorithm capability to find adequate solutions with the initial priority scheme. In the real cases, the planner will be run several times, changing the assigned arm priorities on each run, before a given set of targets would be declared as failed.

8.1 *Scenario in Detail*

Here, we provide full details about a particular scenario. In Fig. 11, the initial and final positions of the arms can be appreciated. During their motions, the arms were exposed to multiple occurrences of all identified collision conflicts that can appear in MIRADAS. Respectively, they are: (i) goal conflicts when the path of an arm passes by the goal location of another one and (ii) en-route conflicts when two paths intersect with each other, but the common point is not the goal location of any of them.



(a)



(b)

Fig. 11 Typical scenario presenting en-route and goal collision conflicts. (a) Top view of the MXS system bench with the 12 arms located at their initial positions. They are numbered, starting from Arm$_0$, counterclockwise. The arms are at their default park locations. (b) Detail of the MXS system with the arms located at their final placements. The numbered black stars at the center of the figure are targets.

We executed a single run of our algorithm with an initial priority policy. Here, as in the previous simulations, arms with larger travel times were given higher priorities. A successful motion plan for all arms was computed in 16.14 s on a 3.2 GHz 2-Core Intel i5 with 4 GB of RAM, as can be seen in Table 1, requiring a total of 75,684 collision checks. As expected, the computation time for each arm trajectory depends on the number of collision checks performed. A trajectory was returned almost immediately for arm 6, the one with the highest priority, since the dynamic environment was empty. However, for arm 10, more than 15,000 checks were needed, yielding a running time

Table 1 Running time and collision checks.

Arm	Priority	Running time (s)	Num. col. checks
6	12	0.002	0
4	11	0.283	1142
5	10	0.616	2164
9	9	0.723	3186
3	8	0.901	4148
8	7	1.401	6103
10	6	3.623	15717
7	5	1.403	6974
2	4	1.548	7816
11	3	1.911	8657
0	2	1.949	9498
1	1	1.780	10279
Total time	...	16.140	...
Average time	...	1.345	...
Total col. check	...	...	75684
Average col. check	...	...	6307

of around 3.6 s. The number of alternative paths present in each arm roadmap was proven to be sufficient for this scenario, accurately representing, therefore, the arms C-space connectivity.

The trajectories found for each arm are given in Table 2. The MXS system arms, their priorities, as well as their estimated travel times, are respectively, shown in column arm, priority, and est. trav. To the right of those columns, the table presents three different groups. The first of them, named accepted traj., refers to the collision-free trajectories returned by our method. The second and the third ones, opt. traj. and roadmap, respectively, show the duration of the optimal trajectory (in Δt units) and information about the paths in each arm's roadmap. The column labeled as path specifies the route followed by each arm.

Arm 6, the one expected to arrive last at the destination, was the first for which a trajectory was computed. Its arrival time, as shown in Table 2, was 19 simulation time steps Δt (114 s), where the optimal trajectory would have taken 18 Δt. For this arm, three different paths with minimal estimated travel time were present in its roadmap. The algorithm randomly selected one of them since that is the normal behavior in such cases, as indicated in Sec. 6. The path first explored comprised two linear segments in the arm's C-space. The associated trajectory had only a single start–stop cycle, performed in the intermediate point A (see field path in Table 2), and during the execution of each segment, there was no any additional stop ($n\Delta t$ stop = 0). The C-space paths followed by this and all other arms as well as the path followed by the centre of each of the pickoff mirrors can be visualized in Figs. 12(a) and 12(b), respectively.

For arm 10, a safe direct trajectory linking the initial and final configurations was found. This single motion trajectory can be seen in Fig. 12(b), where the pickoff mirror trajectory does not present, in contrast with arm 6, any discontinuity. The arm trajectory took a total of 16 Δt, during 2 of which the arm was stopped. These two Δt were carried out in the two first simulation time steps as no start–stop cycles were generated.

Table 2 Trajectories found.

			Accepted traj.					Optim. traj.	Roadmap		
Arm	Priority	Est. trav. (s)	Path	$n\Delta t$	$n\Delta t$ go	$n\Delta t$ stop	nStart–Stop	$n\Delta t$	Total	Minimum $n\Delta t$	Average $n\Delta t$
6	12	107.75	Start-A-end	19	19	0	1	18	32	19	21.34
4	11	95.24	Start-B-end	17	17	0	1	16	33	16	17.88
5	10	92.86	Start-C-end	17	17	0	1	16	33	16	18
9	9	85.29	Start-D-end	16	16	0	1	15	33	15	18.50
3	8	84.46	Start-end	15	15	0	0	15	33	15	16.85
8	7	83.34	Start-E-end	16	15	1	1	14	33	14	17.51
10	6	82.42	Start-end	16	14	2	0	14	33	14	16.79
7	5	81.12	Start-end	14	0	0	0	14	34	14	16.67
2	4	79.63	Start-F-end	14	14	0	1	14	34	14	16.38
11	3	78.45	Start-end	14	14	0	0	14	33	14	16.33
0	2	77.55	Start-end	13	13	0	0	13	33	13	16.78
1	1	63.82	Start-G-end	12	12	0	1	11	33	11	15.82

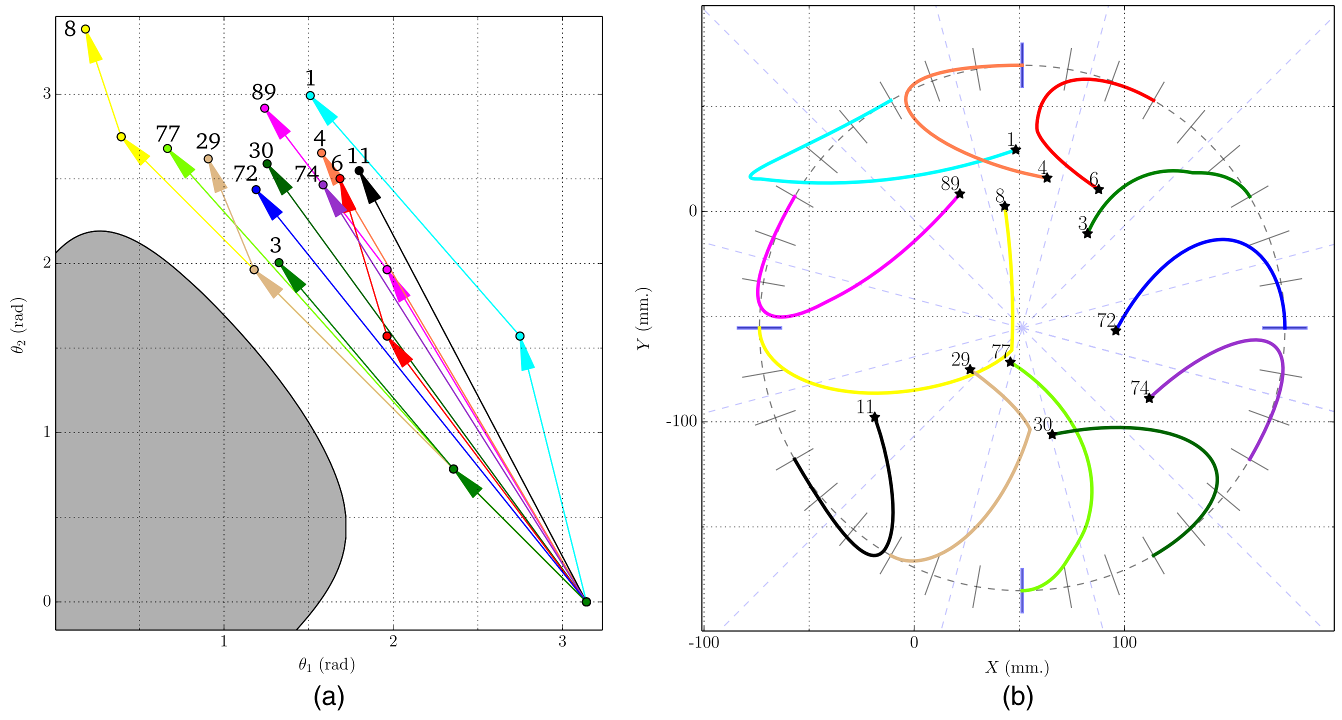


Fig. 12 Paths in C-space and workspace for the typical scenario. (a) The C-space paths followed by the arms. The numbered dots are the same targets than the ones found in Fig. 11. Arrows show the direction motion. All arms start their motions at the same configuration ($\theta_1 = \pi$ and $\theta_2 = 0$) and finish at their corresponding targets. (b) The paths followed by the centers of the pickoff mirrors represented in the arms workspace, the MXS bench. A few arms such as number eight take detours (discontinuities in their paths) to avoid goal-conflicts.

Consequently, when the rest of arms started their respective motions, arm 10 remained in its initial position during 12 s. Finally, as an example of the versatility of the algorithm, it is worth mentioning the result found for arm 8. For this probe, a path with two segments was computed, yielding a start–stop cycle in the connecting point. During one of the 16 time steps that took the trajectory, the arm was stopped. This stop was performed at the beginning of the second segment, remaining the arm at the initial configuration of that segment during 1 Δt. In Fig. 13, we show the sequence of the different motions performed by each arm.

9 Discussion and Conclusion

In this paper, we have presented a motion planning technique for the MXS probe arms of MIRADAS instrument. The algorithm, given a set of 12 initial and final arm positions, computes the collision-free trajectories to be followed by each arm to reach their destinations. Our method is iterative. It assigns a priority to each arm and in descending order tries to find a safe trajectory for each of them independently. The approach samples the control space of the arms, constructing roadmaps comprising several alternative paths. Then, these roadmaps are systematically explored in a state-time space with the help of a DFS strategy, where the number of changes in arm's velocity is strictly monitored and restricted.

The algorithm uses two different parameters, the principal simulation step (Δt) and the collision check step (Δt_{col}), to perform a discrete time analysis. While the former constraints the duration of the atomic motions an arm can perform, the latter, in practice, controls the number of collision checks to carry out per atomic motion. The decoupling of both concepts utilizing two distinct time steps enables our approach to have an extra level of customization. By adequately tuning both, we can achieve a proper trade-off between branching factor, collision detection, and speed.

We have seen that the computation time of each trajectory is tightly bounded to the number of collision checks. The number of checks, in general, increases in every iteration as there are more trajectories previously planned that need to be checked. Furthermore, the execution time of the proposed algorithm is also dependent on the branching factor, which comes determined by the Δt step. The fewer the value of such step, the higher the maneuverability of our arms. However, an increment in the degree of maneuverability comes always at the expense of also incrementing the running time. On the other hand, travel times of the trajectories found are always expressed in multiples of Δt. Thus, by approaching that step to zero, these times can be compacted more and more. But, as expected, that might also affect negatively on the computation time of the solutions.

It has been also shown that trajectories for a typical MIRADAS scenario were computed in around 16 s. Although it might be considered an excessive execution time, it is not when taking into account the operational procedures of the instrument. The motion plans for the different targets to be observed at night will be determined well in advance during day-time. Therefore, the approach is not constrained by online motion planning requirements. However, algorithm running times might be severely decreased if further actions are taken in two particular directions: first, optimization of the collision-checking routines and second, introducing an adaptive

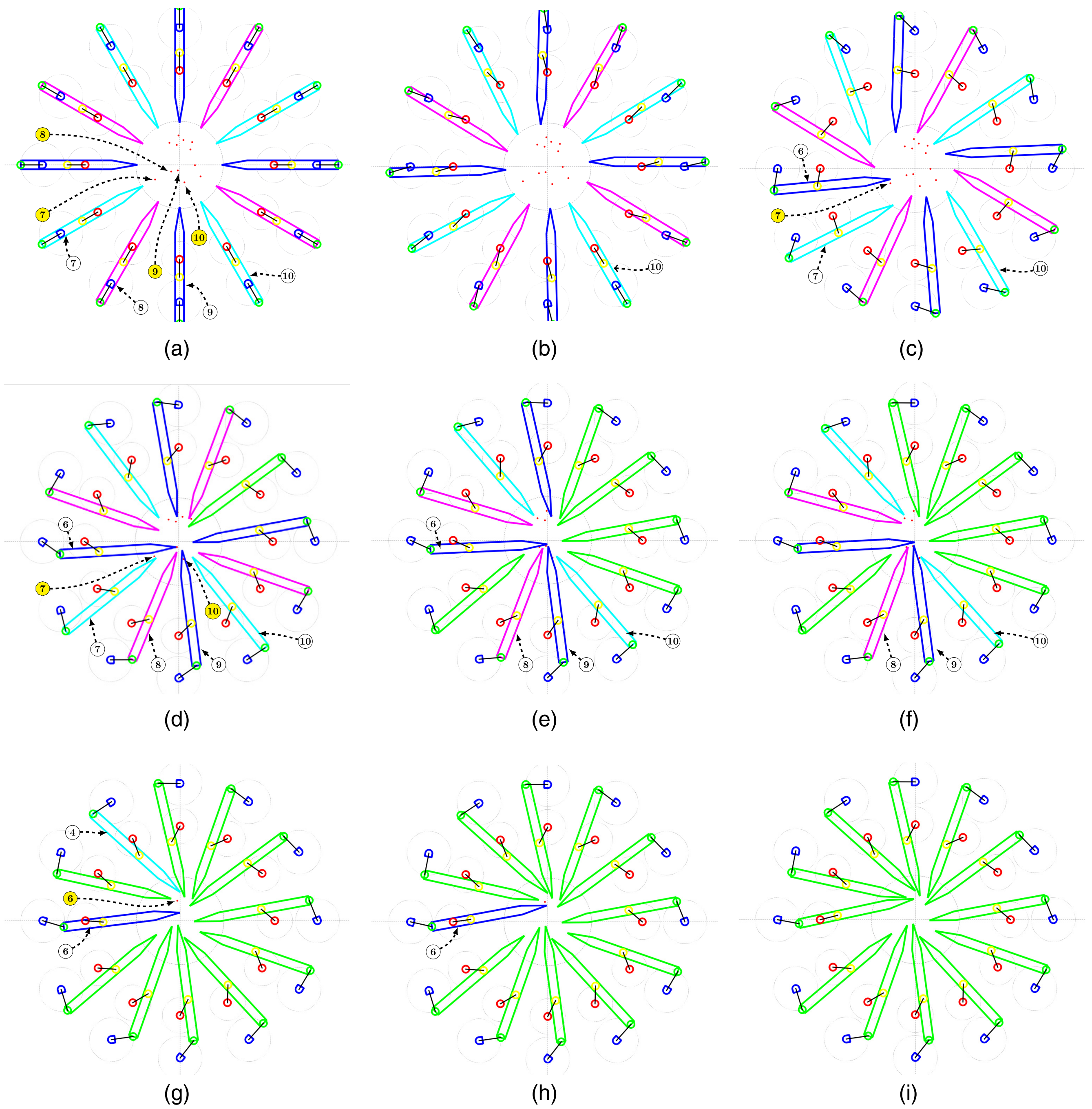


Fig. 13 The sequence of the motions of the 12 arms in the MXS bench. (a) The arms at their start positions. The dots in the center of the figure are the 12 targets. The numbered bold circles point to the target assigned to the corresponding arm. (b) Every probe, except arm 10, starts their motions simultaneously. Arm 10 remains in its initial position. (c) Arm 10 is in motion. Arm 6 is approaching the arm 7 target. As long as arm 6 leaves behind that target before arm 7 reaches it, there will be no goal-conflict. (d) Arm 1 has reached its destination. Arm 6 has left behind the target of arm 7. However, it is invading the arm 8 target and is about to do the same with arm 9 target. Therefore, arm 8 remains motionless. Additionally, arm 9 is over the arm 10 target. (e) More arms (0, 2, 7, 11) keep arriving at their destinations. Arm 6 is leaving behind the targets of arms 8 and 9. Moreover, Arm 9 is leaving also behind arm 10 target. Therefore, Arms 8, 9, and 10 can continue its motions. (f) Arms 8, 9, and 10 are approaching their respective targets. No arm is interfering in their paths, so they are able to safely reach their destinations. (g) All arms, except number 4 and 6, are at their destinations. (h) Arm 6 is about to reach its assigned target. (i) All arms at their final positions.

dynamic collision checking technique.[28] A strategy of that kind would be able to efficiently adapt its resolution to the concrete difficulty of the environment and, at the same time, assuring no collision would be missed even if obstacles are very thin. Moreover, during experimentation, we noticed that an optimization improving performance could also be introduced in the algorithm. Specifically, we found out that in certain situations, a search could be prematurely aborted even before the sequence containing the following states to be visited becomes depleted, saving many unnecessary iterations as well as collision checks. Consequently, once this optimization is implemented, we expect a reduction in the algorithm's execution time.

Simulations for several real science targets show that the algorithm presented with only one pass was able to find in 95% of the cases collision-free trajectories with a configuration time inferior to 120 s. In the scenarios where the planner failed, it was because the motions of a high priority arm prevented a lower priority one reaching its assigned target. This is a situation that can occur in a prioritized motion planner since it is incremental in nature. Therefore, as commented in Sec. 2.1, by default, a replanning with a different arm-priority ordering is performed. In addition, constructing roadmaps with more alternative paths could be also considered. But, unfortunately, these extensions cannot provide completeness to this decoupled approach and, after an array of replannings, it might be that we were not able to determine an acceptable motion planning. In such case, different targets are selected and the process restarts again.

A future step for incrementing the initial success rate of the motion planner will be to include in each roadmap a reserved "garage" location specific to each arm. Circulation along such areas would be only allowed to its respective owners. Additionally, future research also includes, once an arm has reached its goal, opportunistically moving it away to leave room for other arms to pass. In this way, the occasions where higher priority arms prevent motions of the lower ones will be decremented. Finally, another area we plan to explore is using variable arm priorities. That is, in every simulation discrete time step, a different arm will possess the highest priority and, consequently, its motions will not be constrained by the motions of the other arms.

References

1. S. S. Eikenberry et al., "MIRADAS for the Gran Telescopio Canarias: system overview," *Proc. SPIE* **8446**, 844657 (2012).
2. J. Sabater et al., "Target allocation and prioritized motion planning for MIRADAS probe arms," *Proc. SPIE* **9913**, 99132P (2016).
3. W. J. Cook et al., *Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley and Sons, Inc., New York (1998).
4. E. L. Chapin et al., "The infrared imaging spectrograph (IRIS) for TMT: motion planning with collision avoidance for the on-instrument wavefront sensors," *Proc. SPIE* **9913**, 99130T (2016).
5. L. Makarem et al., "Collision avoidance in next-generation fiber positioner robotic systems for large survey spectrographs," *Astron. Astrophys.* **566**, A84 (2014).
6. L. Makarem, J.-P. Kneib, and D. Gillet, "Collision-free coordination of fiber positioners in multi-object spectrographs," *Proc. SPIE* **9913**, 99130V (2016).
7. M. Goodwin et al., "Field target allocation and routing algorithms for Starbugs," *Proc. SPIE* **9152**, 91520S (2014).
8. R. Sharples et al., "Recent progress on the KMOS multi-object integral-field spectrograph for ESO VLT," *Proc. SPIE* **7735**, 773515 (2010).
9. M. Wegner and B. Muschielok, "KARMA: the observation preparation tool for KMOS," *Proc. SPIE* **7019**, 70190T (2008).
10. S. S. Eikenberry et al., "Demonstration of high-performance cryogenic probe arms for deployable IFUs," *Proc. SPIE* **9147**, 91470X (2014).
11. T. Lozano-Pérez, "Spatial planning: a configuration space approach," *IEEE Trans. Comput.* **32**, 108–120 (1983).
12. J. Sabater et al., "Kinematic modeling and path planning for MIRADAS arms," *Proc. SPIE* **9151**, 91515S (2014).
13. P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.* **4**, 100–107 (1968).
14. M. Likhachev et al., "Anytime dynamic A*: an anytime, replanning algorithm," in *Proc. of the 15th Int. Conf. on Automated Planning and Scheduling*, pp. 262–271 (2005).
15. M. Erdmann and T. Lozano-Pérez, "On multiple moving objects," *Algorithmica* **2**(4), 477–521 (1987).
16. S. Buckley, "Fast motion planning for multiple moving robots," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, Vol. 1, pp. 322–326 (1989).
17. M. Bennewitz, W. Burgard, and S. Thrun, "Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots," *Rob. Autom. Syst.* **41**(2–3), 89–99 (2002).
18. D. Silver, "Cooperative pathfinding," in *1st Conf. on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 1, pp. 117–122 (2005).
19. J. P. Van Den Berg and M. H. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Trans. Rob.* **21**(5), 885–897 (2005).
20. L. E. Kavraki et al., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Rob. Autom.* **12**, 566-580 (1996).
21. S. M. Lavalle and J. J. Kuffner Jr., "Rapidly-exploring random trees: progress and prospects," in *4th, Workshop on the Algorithmic Foundations of Robotics; Algorithmic and Computational Robotics, New Directions*, Wellesley, Massachusetts, pp. 293–308 (2000).
22. S. Lindemann and S. LaValle, "Current issues in sampling-based motion planning," in *Robotics Research. The Eleventh Int. Symp.*, P. Dario and R. Chatila, Eds., Springer Tracts in Advanced Robotics, Vol. 15, pp. 36–54, Springer, Berlin, Heidelberg (2005).
23. P. Isto, "Constructing probabilistic roadmaps with powerful local planning and path optimization," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Vol. 3, pp. 2323–2328 (2002).
24. R. Geraerts and M. H. Overmars, "Reachability analysis of sampling based planners," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 404–410 (2005).
25. S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *Int. J. Rob. Res.* **23**(7–8), 673–692 (2004).
26. T. Fraichard, "Trajectory planning in a dynamic workspace: a 'state-time space' approach," *Adv. Rob.* **13**(1), 75–94 (1998).
27. S. LaValle and S. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Trans. Rob. Autom.* **14**, 912–925 (1998).
28. F. Schwarzer, M. Saha, and J. C. Latombe, "Adaptive dynamic collision checking for single and multiple articulated robots in complex environments," *IEEE Trans. Rob.* **21**, 338–353 (2005).

Josep Sabater is a researcher at Institut d'Estudis Espacials de Catalunya (IEEC). He received his BS and MS degrees in computer engineering from University Ramon Llull, Barcelona, Spain, in 2000 and 2001, respectively. Additionally, he obtained his MS degree in electronics from Universitat de Barcelona in 2010. He is currently developing the control system for MIRADAS instrument for Gran Telescopio Canarias. His current research interests include motion planning, robotics, control systems, and artificial intelligence.

Santiago Torres is an assistant professor at Universidad de La Laguna. At the same university, he received his BS and MS degrees in physics in 1998 and 2000, respectively, and his BS degree in electronic engineering in 2005. In 2008, he received his PhD from

Universidad de La Laguna. He is the author of more than 50 journal and conference papers. His current research interests include robotics, control of anesthesia process, and automation.

Francisco Garzón is a full professor at the Department of Astrophysics from Universidad de La Laguna. After receiving his degree in physics from Universidad Complutense de Madrid in 1982, he obtained his PhD in 1987 from Universidad de La Laguna. He is the author of more than 200 papers and has supervised 13 PhD theses. His research interests lie in the field of galactic astronomy and astronomical instrumentation.

José M. Gómez is an associated professor at Universitat de Barcelona. He is also member of the Institute of Cosmos Sciences (ICC) of University of Barcelona. He collaborates in the development of MIRADAS for Gran Telescopio de Canarias. His research is focused on the development of ground based and space borne instrumentation for telescopes.