ME5402/EE5106R Project

Advanced Robotics

Academic Year 2016-2017

Nicolai Domingo Nielsen - Paul-Edouard Sarlin

A0164015R - A0153124U

June 17, 2017

Abstract

As technology breakthroughs have lowered the cost of electromechanical parts and computing systems, industrial robotics has recently attracted a particular attention with promising applications that are no more limited to assembly chains. Here we address the problem of autonomous trajectory planning for aerial work platforms, which are traditionally operated by human technicians but whose safety and efficiency would largely benefit from automation. We propose a novel approach that combines several optimal algorithms into an elegant, efficient and highly generic motion planner which is able to handle manipulators and obstacles in arbitrary configurations. We design a simple controller and perform extensive and realistic simulations, whose results are thoroughly analysed, subsequently validating our implementation and showing the significant promise of this tool for a wide set of highly impacting applications.

Contents

1 Introduction

An aerial work platform (AWP) is a mechanical device used to provide access for people or equipment to hardly accessible areas or equipments. They are often used for maintenance, emergency access or construction purposes, and are therefore commonly mounted on a vehicle. AWP are normally driven by hydraulics, which can be powered by diesel, gasoline or electric motors. Due to its large scale and critical missions, it can be difficult and hazardous for a human operator to precisely control an AWP. We thus study the use of algorithms and intelligent systems to plan and control the exact motion of such a platform.

The motion planning problem can be expressed as finding the appropriate sequence of control inputs to be fed into the AWP controller, based on a priori knowledge of the environment, here assumed static, such that it reaches a given goal position starting from an initial state. Conceptually, the system should find a suitable, safe and elegant trajectory in its environment, allowing both avoidance of obstacles and eventually the reach of the goal within an optimal time. The trajectory will thereafter be referred as the coupled time history of position, velocity and acceleration for each joint.

This problem has been extensively studied by the robotics community and, as an AWP can be directly reduced to a robotic manipulator, various existing solutions can be selected and applied to our specific task. We will start by performing a literature review and briefly analysing commonly used methods. A model of the robot will then be precisely derived. This will allow the design, implementation and analysis of a complete motion planning solution, whose performances will be thoroughly assessed using a realistic simulation of a simple controller. *As a complete implementation with graphical interface is provided, the reader is strongly encouraged to follow along and experiment with it, enabling a deeper understanding of the concepts that will be subsequently presented.*

Figure 1: A simple AWP mounted on a truck.

2 Literature review

Although the motion planning problem may, at a first glance, seem trivial and straightforward for a human, it requires in fact a complex and multi-level behaviour. Several approaches have been explored by robotics researchers and are summarized here.

Most approaches generate a trajectory in the *configuration space* (C-space) of the manipulator, which corresponds to the space of all the configurations of the robot, i.e. the possible joints positions, as opposed to the *workspace* (W-space), which corresponds to the space of the end effector positions. The movement cannot be computed from the W-space, as it does not embed any information about collisions of links with other links or obstacles. Configurations that avoid these collisions form the free C-space, from which the final motion should be drawn, with associated velocity and acceleration profiles. This has the additional advantage that the planner does not have to deal with singularities.

The motion planner is commonly divided into two separate tasks [1], namely *finding the geometric path* and *generating the trajectory*, that can be separately implemented and it is easier for a computer to handle and to solve [2]. The path finding sub-problem consists in computing a set of waypoints in the free C-space, such that, once joined up, they constitute a collision-free short piecewise linear path between the initial and final positions, i.e. a sequence of configurations that lead the robot from one position to the other. From this can be determined a trajectory, by deriving the position, velocity and acceleration of all joints as smooth functions of time. The trajectory optimizes some performance metric while satisfying a set of constraints derived from the dynamic model

2.1 Geometric pathfinding

One simple and popular solution is to discretise the C-space in a grid composed of square cells that are interpreted as nodes of a graph and that carry an internal state, either collision-free or not. Edges link 4- or 8-connected cells together. The shortest path, composed of connected free nodes, is then computed by a standard graph-traversal algorithm, such as Dijkstra [3] or A* [4]. The optimality of such a path however heavily depends on the resolution of the grid, on which depend the length of the path, the computation time and the possible headings of the path, as multiples of a base angle.

Alternatively, one can try to only select nodes that exhibits specific features of the C-space, thus reducing their numbers and the redundancy. As such, generalised Voronoi diagrams [5] or trapezoidal roadmaps [6] with fixed interval allow an efficient partitioning of the space, but only under specific geometric conditions and low-dimensional spaces.

Recently, probabilistic sampling-based algorithms have been largely considered for a wide range of applications. As such, Probabilistic roadmaps [7] randomly sample nodes according to connectivity criteria, while Rapidly-exploring random tree [8], and its op-

timal version RRT* [9], combine sampling and searching, incrementally grow branches towards unsearched parts of the space. These methods work well in high-dimensional spaces, but are not able to declare that no collision-free path exists.

Some alternatives do not rely on graph traversal, but instead on mathematical heuristics. Artificial Potential Fields, for example, model the state as particle influenced by potential fields, both attracted the goal and repulsed by obstacles. They are relatively computationally efficient, but can easily get trapped in local minima, especially in complex environments.

2.2 Trajectory generation

Trajectory planning and optimization can be open-loop, only assuming a set of predefined and fixed constraints, or closed-loop, i.e. taking into account the dynamic model of the robot as well as its response, either on-line or off-line using realistic simulations [10]. The closed-loop methods are very powerful as they allow to perfectly take into account the differential constraints, and are especially important for non-holonomic models, which is not really relevant in our case. They are moreover mathematically complex, involving elaborate implementations of algorithms such as dynamic programming or convex optimization [11].

Simpler methods are often based on the fitting of continuous and smooth curves on the generated waypoints of the path. Different types of splines, quadratic functions or linear segments can be used. Predefined position, velocity, and acceleration profiles can then be mapped along the curve in an open-loop fashion, and be possibly adapted to meet the differential constraints using the dynamic model, as extensively described in [2]. Alternatively, profiles might be independently defined for each individual joint and eventually synchronised and combined into a multi-dimensional path, depending on the type of constraints.

3 Modelling

Before addressing the core problem of motion planning, it is necessary to clearly specify the characteristics of the robot that will be studied as well as all the associated assumptions. We separate here the kinematic modelling, that relates together the configuration space and the workspace, from the dynamic modelling, required for a realistic simulation of the manipulator and its controller.

The AWP from Figure 1 is simplified to a model of a 2-link manipulator as shown in Figure 2. For link $i = 1, 2$, θ_i is the inclination angle of each joint, m_i denotes the masses which are lumped in the middle of each link, l_i are the lengths of each link and l_{ci} is the distance from joint i to the center of mass of the link. For link 1, $l_{c1} = \frac{l_1}{2}$ but since the end effectors mass m_E will be included, the center of mass for link 2 will be located at the distance from joint 2, $l_{c2} = \frac{m_2 \cdot \frac{l_2}{2} + m_E \cdot l_2}{m_2 + m_E}$.

I_i is the moment of inertia about axes coming out of the page passing through the centers of mass, and τ_i is the driving torques at the joints. The links are assumed to be rigid bodies and the motors at each joint are assumed to be ideal on which direct and instantaneous control on the torques will be applied.

The BOB-LIFT GK16AS AWP is chosen as the real world example of the robot system to be modelled and a schematic drawing of it can be seen in Appendix A [12]. The dimensions to be used for the AWP are given in Table 1:

Parameters	**Description**	**Value**
m_E	Mass of the end effector	150 kg
m_1	Total mass of link 1	2041 kg
m_2	Total mass of link 2	3830 kg
l_1	Length of link 1	6.5 m
l_2	Length of link 1	7.5 m
I_1	Moment of inertia of link 1	$3.26 \cdot 10^{-4}$ kg$\cdot$ m^2
I_2	Moment of inertia of link 2	$1.33 \cdot 10^{-4}$ kg$\cdot$ m^2
A_1	Cross sectional area of link 1	0.0625 m^2
A_2	Cross sectional area of link 2	0.0400 m^2

Table 1: AWP real values.

3.1 Kinematic model

In order to analyse the motion of the particles in the system, a kinematics model is presented. The forward and inverse kinematics of the model will respectively show the position of the end effector and the position of the joints, by converting between the workspace and the configuration space presented in earlier.

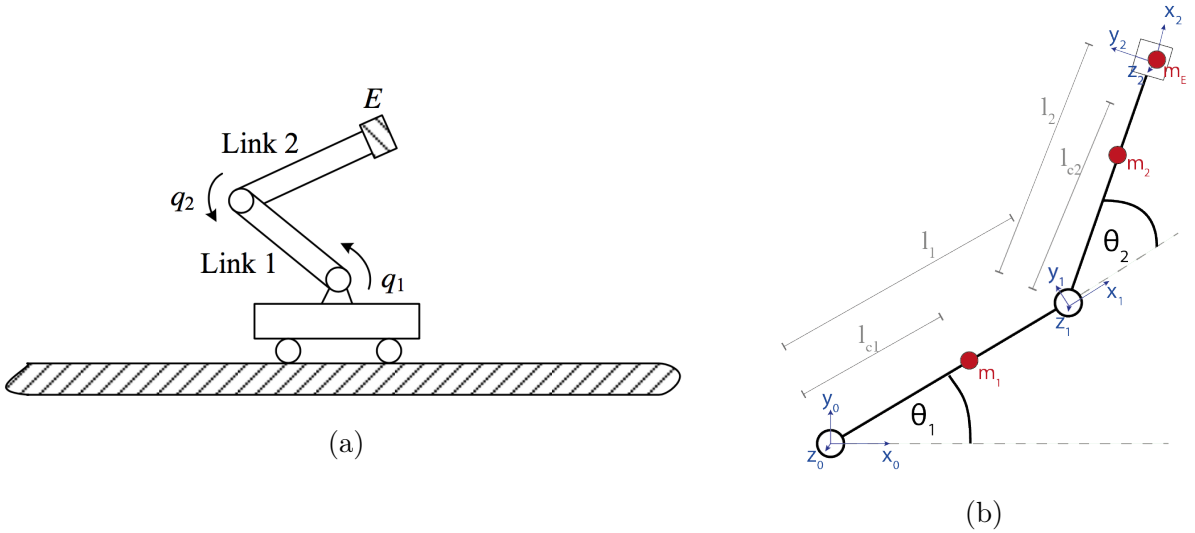


Figure 2: Simplified model diagrams

3.1.1 Denavit–Hartenberg model

The Denavit-Hartenberg parameters of the robotic arm are shown in Table 2. With this convention, the kinematic overview of the system is made clear based on these four parameters, describing the links and the connection to neighbouring links. θ_i is the joint variable, and the other three parameters are fixed link parameters. The Denavit-Hartenberg method is a systematic approach, which allows constructing the forward kinematics by the individual coordinate transformations and it can be applied to any open kinematic chain.

Link i	θ_i	d_i	α_i	a_i
1	θ_1	0	0	l_1
2	θ_2	0	0	l_2

Table 2: Denavit Hartenberg convention

3.1.2 Forward and inverse kinematics

Forward kinematics The forward kinematics relation for the 2-link manipulator, describing the position (x, y) of the end effector is shown in Equation 1.

$$\begin{cases} x &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ y &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \end{cases} \tag{1}$$

Inverse kinematics The inverse kinematics will show the configuration space, being the possible sets of joint angles for a certain position of the end effector. Compared to the forward kinematics, the inverse kinematics is not as simple because of the nonlinearity of the system, so the solutions are not always easy to find and multiple solutions can arise [13].

First θ_2 will be found, and the forward kinematic equation is used as follows, where c_2 gets defined:

$$x^2 + y^2 = l_1^2 + l_2^2 + 2 l_1 l_2 \cos(\theta_2) \quad \Rightarrow \quad c_2 = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

And the existence of solution is given by the condition:

$$-1 \leq \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2} \leq 1$$

And s_2 is set to $s_2 = \pm\sqrt{1 - \cos(\theta_2)^2}$ and $\theta_2 = \text{Atan2}(s_2, c_2)$.

For θ_1, inserting θ_2 into the forward kinematics Equations 1, yields a system of 2 equations with 2 unknowns $\sin(\theta_1)$ and $\cos(\theta_1)$ and the solution is:

$$\begin{cases} x = l_1 \cos(\theta_1) + l_2(\cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2)) \\ y = l_1 \sin(\theta_1) + l_2(\sin(\theta_1)\cos(\theta_2) - \cos(\theta_1)\sin(\theta_2)) \end{cases} \tag{2}$$

Defining $\cos(\theta_1) = c_1$ and $\sin(\theta_1) = s_1$ yields:

$$s_1 = \frac{(l_1 + l_2 cos(\theta_2))y - l_2 \sin(\theta_2)x}{x^2 + y^2}$$

$$c_1 = \frac{(l_1 + l_2 cos(\theta_2))x + l_2 \sin(\theta_2)y}{x^2 + y^2}$$

So, $\theta_1 = \text{Atan2}(s_1, c_1)$ and from here, the joint angles θ can be computed based on the end effectors coordinates (x, y).

3.2 Dynamic model

First of all, the kinetic energy is computed, using the Jacobian matrices [14] as defined below, for respectively link 1 and 2:

$$Jv_{c1} = \begin{bmatrix} -l_{c1}sin(\theta_1) & 0 \\ l_{c1}cos(\theta_1) & 0 \end{bmatrix} \tag{3}$$

$$Jv_{c2} = \begin{bmatrix} -l_1sin(\theta_1) - l_{c2}sin(\theta_1+\theta_2) & -l_{c2}sin(\theta_1+\theta_2) \\ l_1cos(\theta_1) + l_{c2}cos(\theta_1+\theta_2) & l_{c2}cos(\theta_1+\theta_2) \end{bmatrix} \tag{4}$$

And the velocity terms are described as:

$$v_{c1} = Jv_{c1} \cdot \dot{\theta_1} \quad \text{and} \quad v_{c2} = Jv_{c2} \cdot \dot{\theta_2} \tag{5}$$

The translational part of kinetic energy can then be expressed as:

$$K = \frac{1}{2}m_1 \cdot v_{c1}^T v_{c1} + \frac{1}{2}m_2 \cdot v_{c2}^T v_{c2} \tag{6}$$

The rotational kinetic energy needed for including the inertia tensor for the manipulator is defined as:

$$\frac{1}{2}\dot{q}^T \left\{ I_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + I_2 \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right\} \dot{q} \tag{7}$$

And adding the matrices in 6 and 7 yields the inertia matrix $D(\theta)$:

$$D(\theta) = m_1 \cdot Jv_{c1}^T \cdot Jv_{c1} + m_2 \cdot Jv_{c2}^T \cdot Jv_{c2} \begin{bmatrix} I_1 + I_2 & I_2 \\ I_2 & I_2 \end{bmatrix} \tag{8}$$

After the calculations and simplifications of the formulas, and using the standard trigonometric conversions ($\cos^2(\theta) + \sin^2(\theta) = 1$ and $\cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2) = \cos(\theta_1 - \theta_2)$) the inertia matrix will be:

$$D(\theta) = \begin{bmatrix} m_1 \cdot l_{c1}^2 + m_2(l_1^2 + l_{c2}^2 + 2 \cdot l_1 \cdot l_{c2}cos(\theta_2)) + I_1 + I_2 & m_2(l_{c2}^2 + l_1 \cdot l_{c2} \cdot cos(\theta_2)) + I_2 \\ m_2(l_{c2}^2 + l_1 \cdot l_{c2} \cdot cos(\theta_2)) + I_2 & m_2 \cdot l_{c2}^2 + I_2 \end{bmatrix} \tag{9}$$

Now the Christoffel symbols can be derived from the 4 different elements in $D(\theta)$, as

following the procedure:

$$\begin{aligned}
c_{111} &= \frac{1}{2}\frac{\partial d_{11}}{\partial \theta_1} = 0 \\
c_{121} = c_{211} &= \frac{1}{2}\frac{\partial d_{11}}{\partial \theta_2} = -m_2 l_1 l_{c2} sin(\theta_2) \\
c_{221} &= \frac{\partial d_{12}}{\partial \theta_2} - \frac{1}{2}\frac{\partial d_{22}}{\partial \theta_1} = -m_2 l_1 l_{c2} sin(\theta_2) \\
c_{112} &= \frac{\partial d_{21}}{\partial \theta_1} - \frac{1}{2}\frac{\partial d_{11}}{\partial \theta_2} = m_2 l_1 l_{c2} sin(\theta_2) \\
c_{122} = c_{212} &= \frac{1}{2}\frac{\partial d_{22}}{\partial \theta_1} = 0 \\
c_{222} &= \frac{1}{2}\frac{\partial d_{22}}{\partial \theta_2} = 0
\end{aligned} \tag{10}$$

Now the potential energy, which is the sum of the potential energies of the two links, being the mass by gravitational acceleration multiplicated by the height of the center of mass of each link, so:

$$\begin{aligned}
P_1 &= m_1 g l_{c1} sin(\theta_1) \\
P_2 &= m_2 g (l_1 sin(\theta_1) + l_{c2} sin(\theta_1 + \theta_2)) \\
P &= P_1 + P_2
\end{aligned} \tag{11}$$

And the gravitational force vector G(θ) is defined as:

$$G(\theta) = \frac{\partial(P)}{\partial(\theta)} = \begin{bmatrix} (m_1 l_{c1} + m_2 l_1) g \cdot cos(\theta_1) + m_2 l_{c2} g \cdot cos(\theta_1 + \theta_2) \\ m_2 l_{c2} g \cdot cos(\theta_1 + \theta_2) \end{bmatrix} \tag{12}$$

And the C($\theta, \dot{\theta}$) matrix is then, from the Christoffel symbols of Equations 10:

$$C = \begin{bmatrix} -m_2 l_1 l_{c2} sin(\theta)\dot{\theta}_2 & -m_2 l_1 l_{c2} sin(\theta)\dot{\theta}_2 + (-m_2 l_1 l_{c2} sin(\theta)\dot{\theta}_1) \\ m_2 l_1 l_{c2} sin(\theta)\dot{\theta}_1 & 0 \end{bmatrix} \tag{13}$$

And finally, the dynamic equations are described in the following dynamic system:

$$\tau = D(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) \tag{14}$$

Where $\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$, are the torque vectors at respectively joint 1 and 2 of the system.

4 Motion planner

Let us now address the core problem of motion planning. The individual components of our solution will then be presented and analysed. The results will then be thoroughly discussed.

We start by generating the configuration space, in which the geometric path will be computed, using node sampling and graph traversal and pruning, between user-specified start and end points, either in the configuration space or in the workspace. The path, a sequence of waypoints, are eventually turned into continuous position, velocity and acceleration profiles by the trajectory generator using some methods of fitting and smoothing.

4.1 Generating the configuration space

The configuration space a two-dimensional space bounded by the minimum and maximum angle of each joint θ_1 and θ_2. Each point in that space can be either in the free C-space or not. To test its membership, one may perform forward kinematics on $\boldsymbol{\theta}$ and obtain the robot position $\boldsymbol{x}$ in W-space. If there is any overlap between any of the links and any other link or obstacle, then $\boldsymbol{\theta}$ is not a collision-free point.

In our case, there is not risk of inter-link collision for the following reasons: we model links as simple segments, the manipulator has only two DOFs, and the C-space does not wrap-around for any of the joints. In addition to the ground, we decide to introduce obstacles at arbitrary locations in the workspace. For the sake of clarity and simplicity, we constrain these obstacles to circles, although our method is generic and could be applied to arbitrary shapes and environments. For two DOFs, and because we set limits on the joints angles, the collision with the ground is only possible with the end effector. From the forward kinematics equation, as the end effector is modelled as a single point, the collision-free condition is:

$$y(\boldsymbol{\theta}) = l_1 \sin(\theta_1) + l_2 \cos(\theta_1 + \theta_2) > 0 \tag{15}$$

For the obstacles, we discretise the link segments and check, for each point, whether it falls within the circle defined by any obstacle. The resulting collision map is eventually dilated by a predefined distance, resulting in a safety margin in the C-space, which is particularly important for critical applications, although the final path should already not cause any collision.

Figure 3 shows an example of free C-space in white for several obstacles and no dilatation. The C-space has been discretise for this purpose. In white is also shown the reachable W-space, obtained by applying forward kinematics to each cell of the free C-space. Because of the high non-linearity of the transformation, it is difficult to visually relate one to the other.

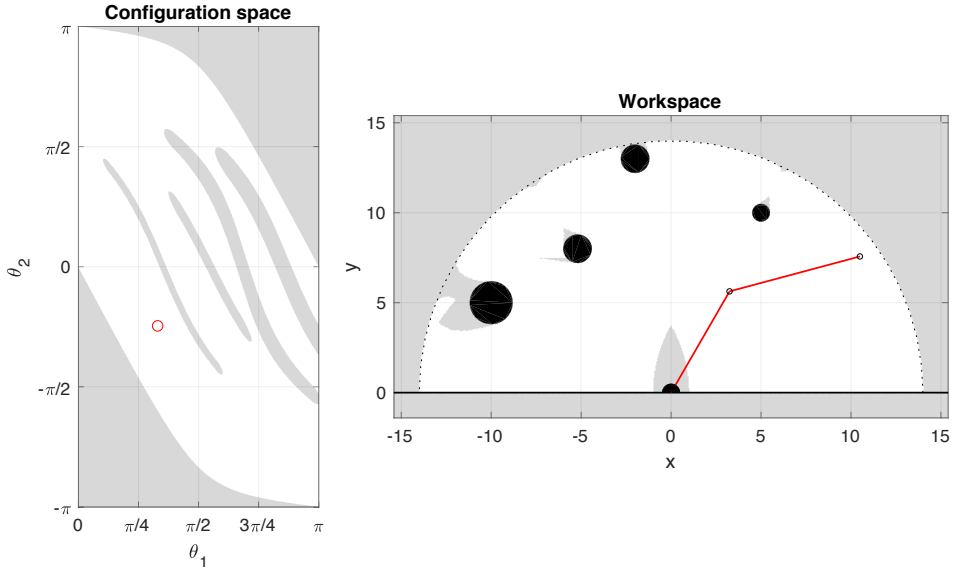


Figure 3: C-space and W-space for a set of obstacles (black disks), manipulator with red links, free C-space and reachable W-space in white.

4.2 Target point

The target point, i.e. the point that the motion planner should reach, can be either set in the C-space or in the W-space. In the former case, the target is unambiguous, as long as the point lies in the free C-space. In the latter case, one has to perform inverse kinematics on (x, y) to obtain (θ_1, θ_2). Due to the kinematic characteristics of the manipulator, there are always two possible sets of joint angles, although they do not necessarily all lie in the free C-space. Even after discarding points that are not free, multiple sets of angles can remain. We perform path finding on each of them and only keep the set corresponding to the shortest path, i.e. which is closer to the initial point.

4.3 Geometric pathfinding

Now that the free C-space and the end points are determined, the geometric path has to be constructed. We decide to generate the nodes with a Probabilistic Roadmap (PRM), which is particularly attractive for its computational efficiency and graph optimality. That path is computed using A* and pruned for a smoother trajectory.

4.3.1 Nodes sampling

A probabilistic roadmap (PRM), relies on the random sampling of nodes and their local connection. The algorithm, introduced in [7], iteratively samples n points from a uniform distribution over the free C-space χ_{free} and adds them to the vertex set V. Each new point

x_i is attempted to be connected to other other vertices in V that are within a distance r. For the connection to be valid, no obstacle should lie on the straight line between the two points, i.e. the line should entirely belong to χ_{free}, and both points should be part of the same connected component. If the connection is successful, the corresponding nodes of the graph are linked with a new edge in E, whose weight corresponds to a predefined cost, here the Euclidian distance between the points. Two parameters, n and r, need to be tuned beforehand. The corresponding pseudocode is shown in Algorithm 1.

Algorithm 1: PRM

Input: number of nodes n, radius of search r, obstacle-free C-space χ_{free}

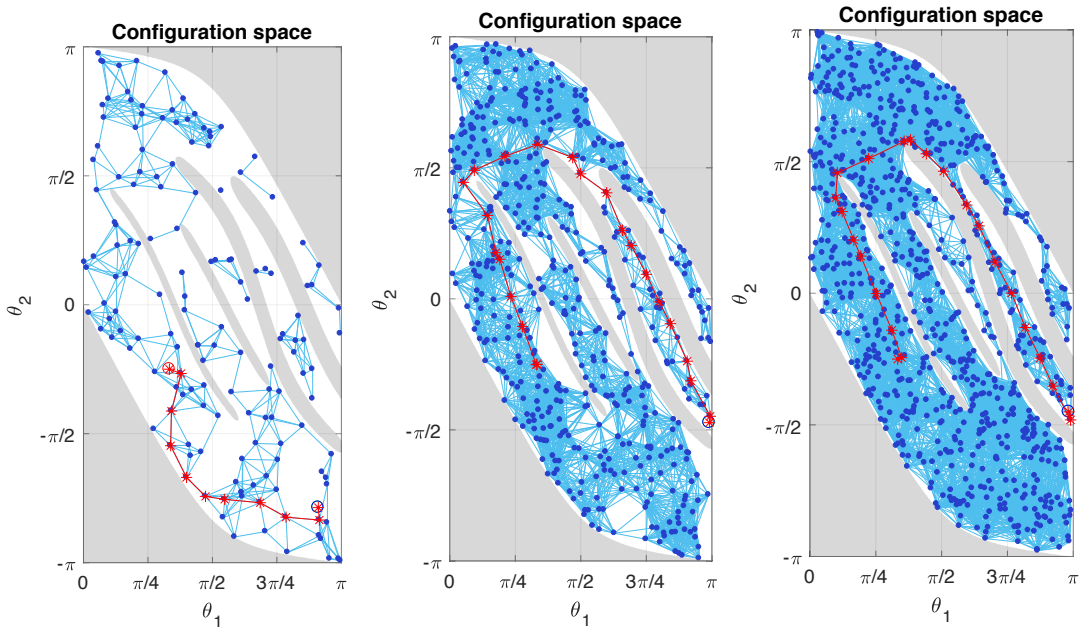
$V \leftarrow \emptyset$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_i \leftarrow \texttt{sample}\,(\chi_{\text{free}})$;
  $U \leftarrow \texttt{within}\,(G = (V, E), x_i, r)$;
  $V \leftarrow V \cup x_i$;
  foreach $u \in U$ **do**
    if x_i *and* u *are not in the same connected component of* $G = (V, E)$ **then**
      if $\texttt{line}\,(x_i, u) \subset \chi_{\text{free}}$ **then** $E \leftarrow E \cup \{(x_i, u), (u, x_i)\}$;
return $G = (V, E)$;

As n increases, the nodes have a higher probability to be inter-connected, and a path is thus more likely to be found. Moreover, it can be proven that PRM monotonously converges towards the shortest path. Higher n and r tend to decrease the length of the path, but increase the computational complexity of the graph search for the path query. These should thus be adjusted to the lowest value possible such that they still lead to a path of acceptable quality. We use a PRM implementation of the MATLAB Robotics Toolbox, with the built-in occupancy grid data structure. The algorithm turned out to be quick enough to easily handle up to 2000 nodes.

4.3.2 Path query

Once the graph is computed, a path-finding algorithm can be used to determine the set of nodes that together constitute the shortest path. The A* algorithm [4] appeared to be an appropriate solution, being both optimal and computationally efficient. Starting from the initial point, it iteratively propagates a move cost, corresponding to the weight of the visited edges, i.e. the Euclidian distance between the nodes. The algorithm always expands first the path that minimizes the total semi-estimated cost f from start to goal.

For each node i, $f(i) = g(i) + h(i)$, where g is the movement cost from start to i, and h is the estimated cost from i to the goal, called the heuristic function and chosen to be the Euclidian distance.



(a) $n = 100$ (b) $n = 500$ (c) $n = 800$

Figure 4: PRM nodes (dark blue disks) and edges (light blue lines), waypoints selected by A* (red asterisks) for arbitrary initial and final points.

Figure 4 shows the resulting nodes and path for $n = 100$, $n = 500$ and $n = 800$. As the size of the graph increases, the path converges to the optimum, but is also composed of more nodes. We note that for a low number of nodes (e.g. $n = 100$), the algorithm is not able to find a path leading to very narrow parts of spaces, but this becomes possible as n grows.

4.3.3 Pruning

When computational complexity is not an issue, increasing the number of nodes can give much better results. However, as mentioned previously, it does also increase the number of nodes that the final path is composed of. This can be an issue for the trajectory generation, producing jerky movements, especially if the nodes are out of alignment.

We therefore decide to remove unnecessary nodes from the path, i.e. to prune the path. For each point of the path p_i, we check if the line to a subsequent point p_j is in the free C-space. If this is the case, we increment j and check again. This is performed for each subsequent point p_j, until the collision is not met, e.g. at point p_k. We then discard

all the points of the path between p_{i+1} and p_{k-2} (included), and start the process again with $i = k - 1$. An example of result is shown in Figure 5. The number of waypoints is successfully reduced, and the piecewise linear path already looks smoother.

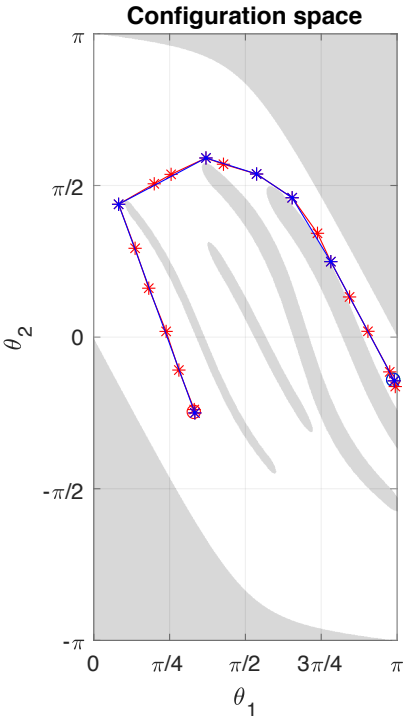


Figure 5: Pruning decreases the number of waypoints from 19 (red) to 7 (blue).

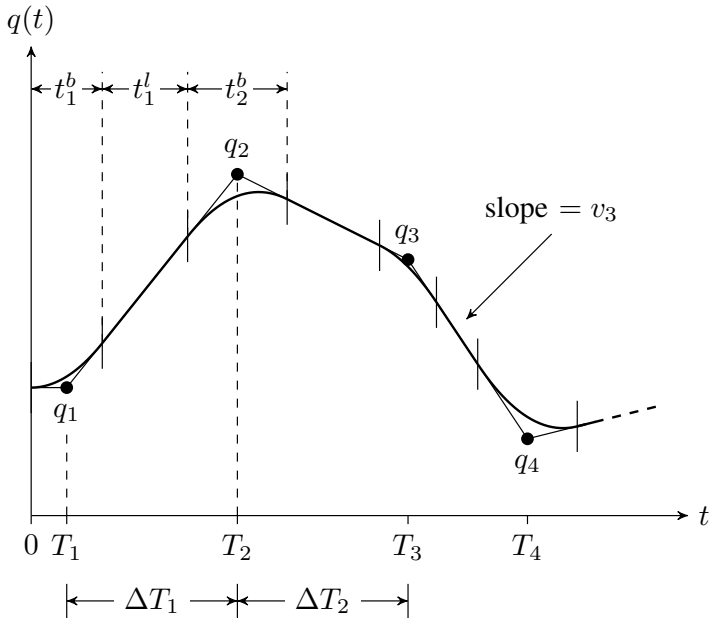


Figure 6: Example of position profile computed using LSPB.

4.4 Trajectory generation

Now that the path and its waypoints are defined, position, velocity and acceleration reference values have to be determined for any point in time. These will be later fed into the controller to make the actual robot move and accomplish the desired task.

4.4.1 Constraints and chosen method

We decide to set constraints on the movement of the joints. Their velocities should be continuous, as it would otherwise require an infinite acceleration, which the motors would not be able to provide, preventing the controller from following the path. Additionally, in order to further take into account the physical limitations of the real system, the absolute values of the velocity and acceleration of each joint should be bounded.

A trivial method would be to come to a stop at every waypoint. This is totally suboptimal from the perspective of time. We thus seek for a more elegant alternative.

The above-defined constraints require the curve to have a continuous second derivative, i.e. curvature. Polynomials and splines easily satisfy these conditions, provided that their order is sufficiently high. It can be however mathematically difficult to independently compute them for each individual joint such as the velocity and accelerations are as high as possible but below the limits, while ensuring that the waypoints are reached at the same time for each joint. Linear segments are much simpler to compute, and can be made smooth by adding a parabolic blend at each waypoint. This is the solution that the develop here. One of their key advantage is that they remain close to the linear piecewise path, only diverting at the waypoints, and ensuring that the trajectory does not collide with any obstacle.

4.4.2 Motion law

The main idea behind the linear segments with parabolic blends (LSPB) is introduced in [15]. Conceptually, a constant velocity is assumed between the waypoints, and a trapezoidal profile, i.e. constant acceleration, is used to switch between two subsequent segments, ensuring a smooth movement. An example is shown in Figure 6. However, the initial version of the algorithm computes the polynomial coefficients from predefined timestamps of the waypoints. In our case, the timestamps are unknown. The algorithm should instead try to reach the maximum velocity, and thus compute the minimum duration of each linear segment. Velocity changes should however be coherent with the acceleration limits. We thus implement an alternative, introduced in [16], that automatically chooses durations to generate a smooth, valid trajectory.

We start by computing, for each segment i, the duration ΔT_i corresponding to the maximum time that it takes for all the joint j to travel along it at their respective maximum speed $v^j_{\max}$. We thus minimize the travel time while ensuring that the constraints are met. The, iteratively for each segment, the corresponding velocities v^j_i are computed, as well as the necessary velocity changes $v^j_i - v^j_{i-1}$, and the required durations of the blends t^b_i to perform these changes at the maximum acceleration allowed $a^j_{\max}$. If the blends are overlapping, i.e. if $t^l_i < 0$ in Figure 6, then the durations of the adjacent segments T_i and T_{i+1} are increased by a factor f and the velocities of the impacted segments are recomputed.

Figure 7 shows the movement of the robot for a simple obstacles configuration. In the W-space, the trajectory of the end effector is shown in green, while the initial, final, and some intermediate manipulator configurations are shown in red, blue and black respectively. In the C-space, we show the smooth trajectory as a solid blue line. As the final position was set in the W-space, the green circle represents the second end configuration that resulted from the inverse kinematics, and which was not selected because located in

a collision area. The position, velocity and acceleration profiles of the joints are shown in Figure 8. Additionally, we also perform the instantaneous forward kinematics and show the velocities of the end effector. The maximum velocities and acceleration are $v_{max} = 0.1\,\mathrm{rad\,s^{-1}} = 5.7\,^\circ\,\mathrm{s^{-1}}$ and $a_{max} = 0.005\,\mathrm{rad\,s^{-1}}$ respectively for both joints.

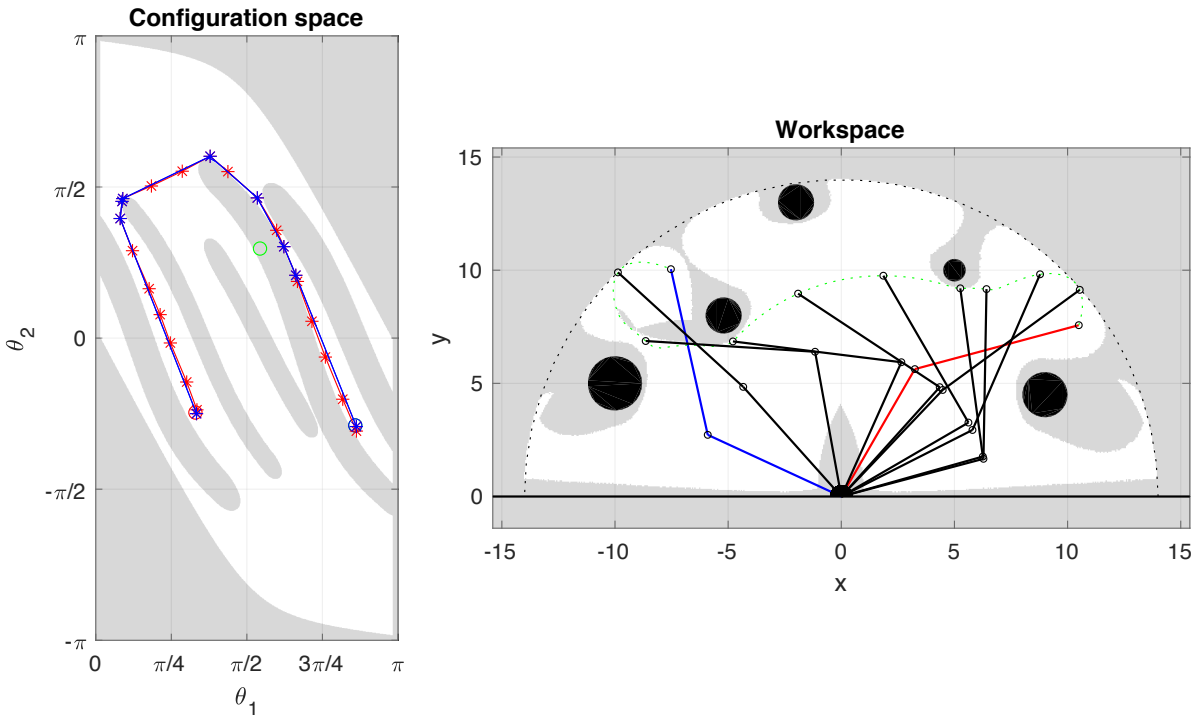


Figure 7: An example of path, with initial, intermediate and final configurations (red, black and blue respectively) and dilatation of the C-space.

We observe that the final trajectory does not exhibit any collisions with obstacles. The position profile is smooth, and the blends can clearly be noticed near the waypoints, shown as black asterisks. The velocities are, as expected, in a trapezoidal shape and never exceed the maximum value, while being sometimes close to it. The acceleration profiles meet the constraints as well. The velocity of the end effector is continuous, while very different form the joint velocity profiles because of the non-linearity of the transformation. It however remains within realistic bounds.

As the results are rather satisfactory, we decide to simulate the system with a simple controller and the previously-derived dynamic equations. This will allow us to show how realistic and elegant our solution is.

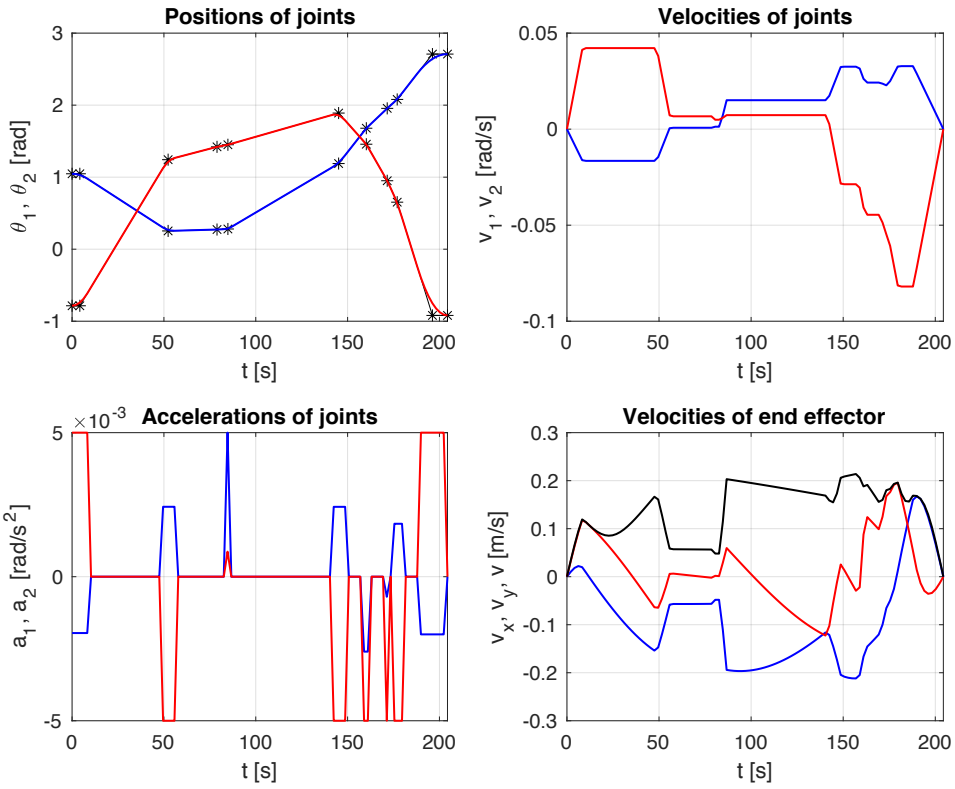


Figure 8: Position, velocities and accelerations for θ_1 (blue) and θ_2 (red), x (blue), y (red) and total (black) end effector velocities.

5 Controller

The kinematic model of a robot assumes a perfect behaviour of a robot, i.e. that all types of joint trajectories are perfectly executed. In the real world this is not the case and uncertainties, the environment and the dynamics of the robot will affect the motion. In order to compensate for these factors, a controller is needed. In this case, the equation of motion for a two link manipulator is, as seen earlier, a non-linear differential equation and therefore the solutions are computed numerically. Due to this non-linear behaviour, an efficient controller is needed for the system in order to move the end effector as accurately as possible.

Different types of controllers can be used, such as PD, PID, computed torque, adaptive or neural network controllers. As our goal is to show that the generated trajectory is smooth and realistic, we focus on a simple PID controller with gravity compensation and observe that this is sufficient for ensuring a collision-free movement.

5.1 Design

The PID controller is based on the dynamic system presented earlier:

$$\tau = D(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) \tag{16}$$

Isolating $\ddot{\theta}$ yields:

$$\ddot{\theta} = D(\theta)^{-1}[-C(\theta, \dot{\theta}) - G(\theta)] + \hat{\tau} \tag{17}$$

Where $\hat{\tau}$ is defined as $\hat{\tau} = D(\theta)^{-1}\tau \iff \tau = D(\theta)\hat{\tau}$.

Now these new (non-physical) inputs are described where $\hat{\tau}$ is defined as $\hat{\tau} = \begin{bmatrix} u_1 \\ u_1 \end{bmatrix}$

And the physical torques to the system are defined as $u_{\theta 1}$ and $u_{\theta 2}$:

$$\begin{bmatrix} u_{\theta 1} \\ u_{\theta 2} \end{bmatrix} = D(\theta) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

The error signals are defined as $e(\theta_1) = \theta_{1d} - \theta_1$ and $e(\theta_2) = \theta_{2d} - \theta_2$. Where θ_1 and θ_2 will be the computed joint angles after the control, and θ_{1d} and θ_{2d} are the desired joint angles computed by the optimal trajectory generator.

The initial positions of the manipulator are chosen to be $\theta_{init} = \begin{bmatrix} \pi/3 \\ -\pi/4 \end{bmatrix}$

The structure of the chosen controller follows the structure of a simple PID:

$$u = \ddot{\theta} + K_p e + K_D \dot{e} + K_I \int e(\theta_) + D(\theta)^{-1} G(\theta) \tag{18}$$

Since the gravity terms showed instability, especially in the beginning of the trajectory, causing static position errors, the gravity compensation term $\hat{G}$ was added to the control

system. The above equation for u is then translated into the 2 link manipulator system as:

$$\begin{aligned} u_1 &= \ddot{\theta} + K_{p1}(\theta_{1d} - \theta_1) + K_{D1}(\dot{\theta_{1d}} - \dot{\theta_1}) + K_I \int e(\theta_1)dt + D(\theta)^{-1}G(\theta)\big|_1 \\ u_2 &= \ddot{\theta} + K_{p2}(\theta_{2d} - \theta_2) + K_{D1}(\dot{\theta_{2d}} - \dot{\theta_2}) + K_I \int e(\theta_2)dt + D(\theta)^{-1}G(\theta)\big|_2 \end{aligned} \tag{19}$$

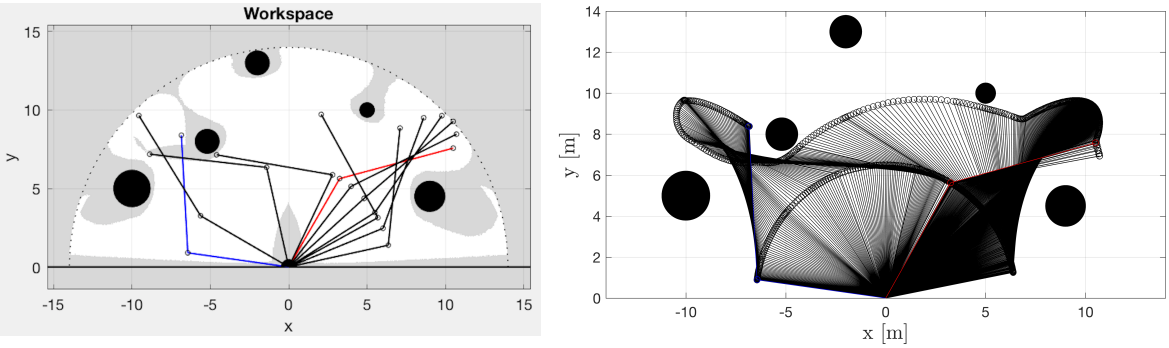
And the complete system with the above mentioned control terms is:

$$\ddot{\theta} = D(\theta)^{-1}[-C(\theta, \dot{\theta}) - G(\theta)] + \hat{\tau} \tag{20}$$

5.2 Results

The above method is implemented in MATLAB using its `ode45` Runge-Kutta solver. The trajectory path with its θ, $\dot{\theta}$ and $\ddot{\theta}$ values are firstly computed and used as inputs values for the control system.

The main trajectory case studied, is the one which can be seen in Figure 9a. In this case, and as shown in Figure 12a, it is a trajectory based on obstacle avoidance where the black dashed line is the desired trajectory and the end effector needs to reach the backside of the circular obstacle located at [-5m, 10m].



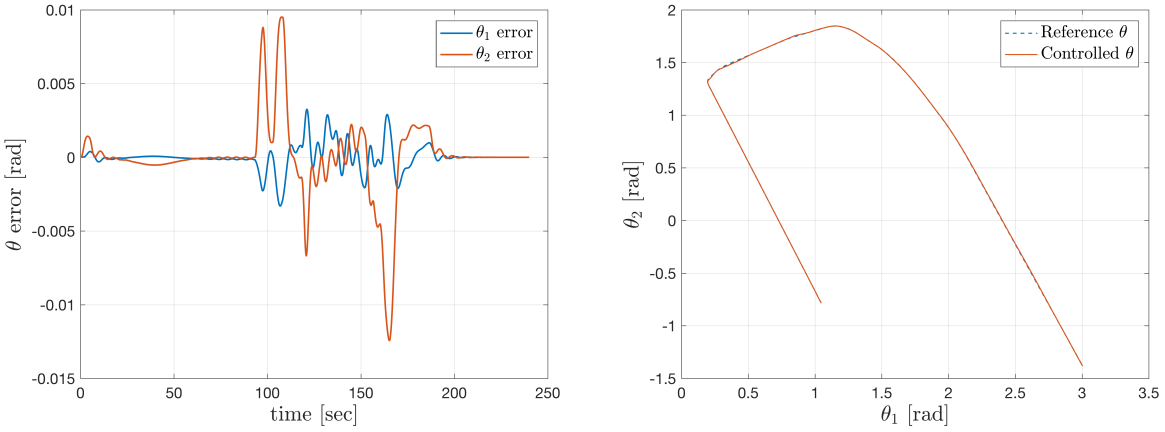
(a) Trajectory with obstacle avoidance

(b) Controlled trajectory with high framerate

Figure 9

For finding appropriate gain values for the proportionality K_p, the derivative K_D and the integration K_I, manual tuning was used and the parameter values showing best results were: $K_{p1} = 17$, $K_{D1} = 9$, $K_{I1} = 11$ for joint 1 and $K_{p2} = 18$, $K_{D2} = 13$, $K_{I2} = 12$ for joint 2.

For lower gain values, the error between the desired θ_d values from the trajectory planning and the computed controlled θ values (e=$\theta_d - \theta$) and the configuration space (θ_1, θ_2) can be seen in Figure 10. The error goes too high, especially for the angle of joint 2, and the θ trajectories are off the desired path due to the low control input gains, but still it is able to follow the path during most of the trajectory which shows that even with low gains, the controller provides acceptable compensations.

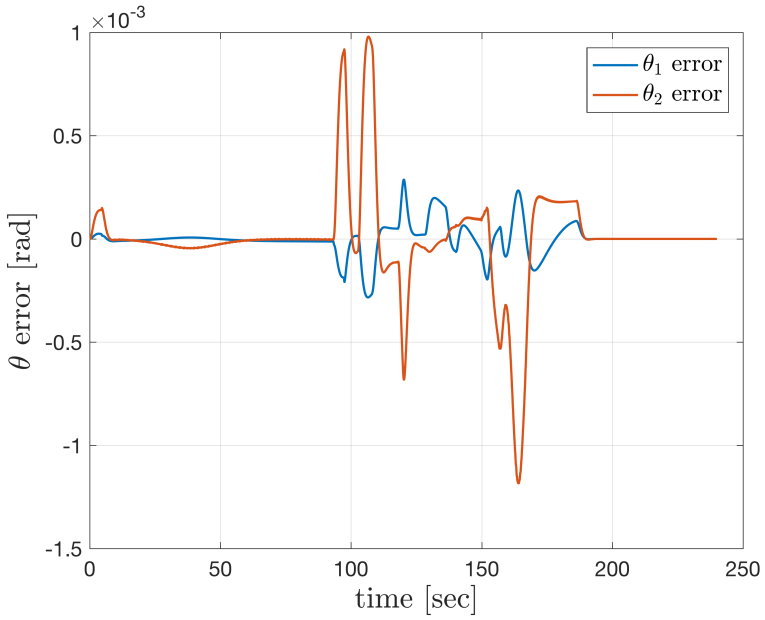


(a) Relatively high e due to low gains

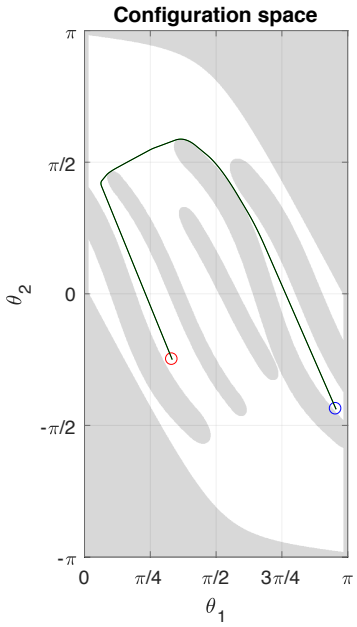
(b) Trajectory following

Figure 10: $K_{p1} = 2$, $K_{D1} = 1$, $K_{I1} = 1$, $K_{p2} = 3$, $K_{D2} = 2$, $K_{I2} = 1$

In Figure 11a the error plot is now shown for the appropriate gain values, and the difference between the desired and the controlled values goes really small compared to earlier. There is though a peak at about t=160 seconds, which is the point where the manipulator has to avoid the obstacle and turn into the allowed workspace. This is referring to the manipulator position where the end effector is located at approx. [-10.6 m, 9.5m] which can be seen in Figure 12a.



(a) Decreased e at tuned gains

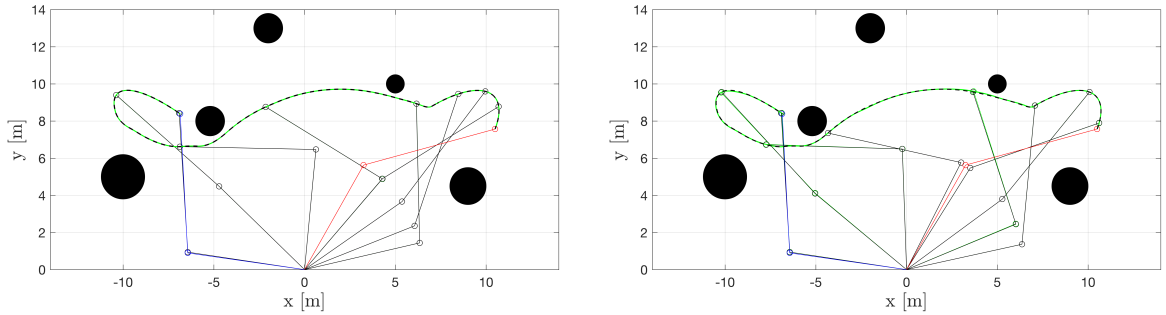


(b) Accurate trajectory following

Figure 11: $K_{p1} = 17$, $K_{D1} = 9$, $K_{I1} = 11$, $K_{p2} = 18$, $K_{D2} = 13$, $K_{I2} = 12$

In `pathGen.m` the motion of the manipulator is plotted in the workspace. The red manipulator is the starting position, the blue manipulator is the final position, the black manipulator is the desired positions at every time interval T and the green manipulator is the computed controlled positions. The filled black circles represent the obstacles to be avoided. It is here important to assure that, with the chosen PID controller, no collision is possible, neither between the end effector and the obstacles nor between any point on the links and the obstacles. This can be seen to be satisfactorily achieved in Figure 9b, where the movement of the manipulator is recorded at a high frame rate for capturing the path of all its links. In Figure 11b, the configuration space is plotted where it can be seen that no collision is happening for the chosen controller and its computed trajectory.

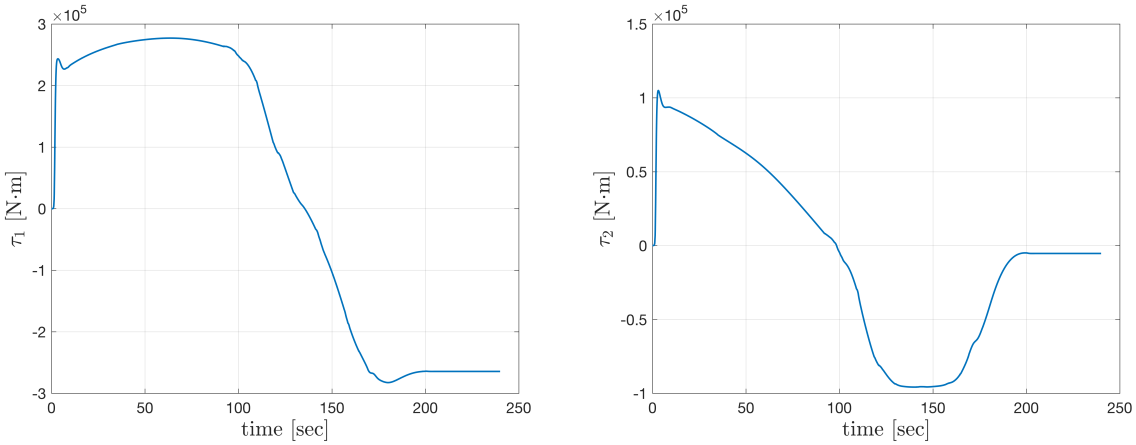
A comparison between the manipulator positions with low gains (same as the ones from Figure 10) and the chosen ones can be seen in Figure 12. The green line shows the controlled end effector positions throughout the trajectory. At this representation, the low gain PID controller seems to have a similar performance compared to the chosen K values, but as shown by the error, the tuned controller performs better overall.



(a) Trajectory following at tuned gains (b) Trajectory following for low gain values

Figure 12

Finally, plots of the torque for joint 1 and 2 are shown in Figure 13. For assuring less aggressive and more realistic manipulator movements, the angular velocity maximum was set to 0.1 rad/s and the angular acceleration to 0.005 rad/s^2. With these values, when joint 1 needs its maximum torque at t=10s, a perfect motor will consume 2.19 kW while the motor joint 2 will consume 2.38 kW, values which are quite low for such motors, but the reason must be the simple assumptions made, neglecting friction and similar factors.



(a) Torque for joint 2 (b) Torque for joint 2

Figure 13: $K_{p1} = 17$, $K_{D1} = 9$, $K_{I1} = 11$, $K_{p2} = 18$, $K_{D2} = 13$, $K_{I2} = 12$

6 A note on optimality

Here have previously shown that our solution successfully meets the requirements, ensuring a realistic behaviour for the robot. Let us now briefly discuss its optimality, that we define here as the minimization of the travel time, while still meeting the constraints. Spacial optimality, i.e. the minimization of the length of the path, is already mentioned, as these are tightly related.

The graph outputted by PRM is supposed to provide a good coverage of the given space if the number of nodes is high enough. This means that the distance between two arbitrary nodes is well approximated by the shortest path through the edges of the graph. As computing power is not a problem, we can generate far more nodes than necessary and thus obtain a graph that correctly represents the spacial configuration of the free C-space.

A* finds the shortest path in the graph. Assuming that it already correctly covers the space, as mentioned above, the path returned by A* is the shortest possible between two given points. Pruning does not affect the spatial optimality of our solution, as it decreases the number of nodes but does not affect the total distance. Thus, we achieve spatial optimality.

LSPB closely follows the path but cuts out the corners, thus only slightly shortening the final path. LSPB tends to maximise the velocity and acceleration along the path, and thus minimises the travel time along a distance that is already the shortest possible. In that sense, our solution can be considered optimal

7 Limitations and possible improvements

Although the optimality of the proposed solution has been previously demonstrated and the project requirements are met, our system still presents several weak points.

While the path finding part seems to perform very well, the main improvement resides in the way the trajectory is generated, as it does not directly take the dynamic limitations of the system into account. Differential constraints, obtained from the dynamic model of the manipulator, should be incorporated into the planner, ideally as an closed loop controller, where the motion planner is aware of the limitations of the controller and thus adjusts the velocity and acceleration of the joints consequently. Discontinuous accelerations should be avoided, as they may excite vibrational modes which potentially can reduce the accuracy and durability of the system.

Secondly, we only considered the constraints on the joints, but not on the end effector. If it should carry people, then higher orders limitations should be implemented, taking into account the acceleration as well as the jerk. This results in an even more complex planner, which would potentially require more computing power than available.

The proposed controller is limited to the tuned gain values, so if unexpected uncertainties or disturbances appear or if a more complex model is implemented, the PID controller can present low robustness to these. And being that an AWP naturally can be exposed to unexpected environmental conditions, this concern is effective.

8 Conclusion

The proposed solution successfully provides an efficient, elegant and generic answer to the problem of AWP motion planning by breaking it into several smaller sub-problems that are individually handled. Points sampled by PRM* are combined by A* to form a piecewise linear path that ensures a collision-free movement and minimises the travel time. To cope with the physical limitations of the robot, a smooth curve is subsequently generated using parabolic blends, resulting in an time-optimal trajectory that yet meets predefined velocity and acceleration constraints. A PID controller with gravity compensation is designed to execute the trajectory. A realistic simulation, based on an extensive dynamic model of the manipulator, allows a fine tuning of the various parameters, such as the controller gains, and shows that our system results in a successful and practically viable avoidance of the obstacles, thus demonstrating the substantial promise of our solution.

These results are evaluated mainly based on the ability of the manipulator to follow the trajectory. Such an AWP is supposed to carry people on the end-effector, for which the control part is important in order to ensure safe velocities and accelerations of the platform. More complex differential constraints, that guarantee safety and durability while remaining optimal, still need to be taken into account for further improvements.

References

[1] A. Lanzutti A. Gasparetto, P. Boscariol and R. Vidoni. Trajectory planning in robotics. Springer Basel, Mathematics in Computer Science, 2012.

[2] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[5] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin. Path planning for mobile robot navigation using voronoi diagram and fast marching. pages 2376–2381, Oct 2006.

[6] Ileana Streinu. Trapezoidal path planning. `http://cs.smith.edu/~streinu/Teaching/Courses/274/Spring98/Projects/Philip/fp/algPath.htm`, 1998.

[7] Lydia Kavraki, Petr Svestka, Jean Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Technical report, Stanford, CA, USA, 1994.

[8] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[9] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[10] C.S.G. Lee K. S. Fu, R.C. Gonzalez. Robotics: Control, sensing, vision, and intelligence. 1987.

[11] David H. Jacobson and David Q Mayne. *Differential dynamic programming*. American Elsevier Pub. Co, 1970.

[12] Bob-lift. Model gk16sa. `https://www.alibaba.com/product-detail/Elevated-Work-Platform-Truck-Mounted-Awp_60304132262.html`.

[13] Chew Chee Meng. Inverse kinematics lecture slides for me5402/ee5106r. 2017.

[14] Seth Hutchinson Mark W. Spong and M. Vidyasaga. *Robot Dynamics and Control*. John Wiley & Sons Inc, New York, NY, USA, second edition edition, 2004.

[15] J.J. Craig. Introduction to robotics, mechanics and control. 3rd Edition, Pearson Prentice Hall, 2005.

[16] Tobias Kunz and Mike Stilman. Turning paths into trajectories using parabolic blends. Technical report, 2011.

Appendices

A BOB-LIFT GK16SA Specifications