Direct Trajectory Generation for Vision-Based Obstacle Avoidance

Roel Pieters, Alejandro Alvarez-Aguirre, Pieter Jonker and Henk Nijmeijer

Abstract—In this paper, a method for direct trajectory generation for robotic manipulators is proposed. The method is specifically designed for obstacle avoidance and can incorporate kinematic constraints into the avoidance motion. In particular, a point-to-point and multi-point trajectory generator is proposed in which different levels of constraints can be established on via- and end-points. The approach is direct in the sense that at every time instant a new trajectory is generated, which allows the trajectory to be changed at runtime. Moreover, when the final conditions of the trajectory are altered such that constraints would be violated, an optimization procedure is employed in order to extend the execution time of the trajectory. This effectively makes the trajectory time-optimal. The approach is experimentally verified with a 7-DOF anthropomorphic manipulator with time-synchronization of multiple trajectories.

I. INTRODUCTION

With the increasing demand of integrating robotics into every day life and industry, safety requirements are still a driving factor. In a human-centred environment, robot motion has to be as smooth as possible and safety has to be guaranteed. This implies a safe replanning of motion when obstacles are detected. As current state-of-the-art approaches differentiate between obstacle avoidance (i.e., path planning) and traditional positioning (i.e., trajectory planning), the problem of avoidance is usually solved by designing a new path. This means that predefined kinematic constraints for the trajectory are not taken into account for obstacle avoidance and only a reactive motion guides the robot away from objects (e.g. potential field, roadmap).

This combination of traditional motion control with direct and online replanning of trajectories is the concept of this paper. In particular, our approach designs a new trajectory at every iteration, even when no obstacle is detected. This enables the generation of a trajectory only for the next state of the motion system based on current state and events. As such, sudden, unexpected actions that need replanning of motion can be taken into account. Direct trajectory design is presented for point-to-point and multi-point positioning, for different levels of constraints. More specifically, a different choice of constraints on (via-)points and on the complete trajectory itself will result in a different motion design. In order to guarantee that motion bounds are not violated, an optimization procedure is employed which regards the extension of the execution time in order to meet predefined kinematic constraints.

A. Related work

Trajectory design for robot motion control is one of the earliest fields of research in robotics. Traditional approaches that are now accepted as standard implementations can be found in many well-known textbooks (see for example [1], [2]). It is well known that path planning [3] and trajectory planning [4] are two different topics. The former considers only the geometry of positioning, while the latter considers time and can thus include constraints on for instance velocity and acceleration. This difference is of importance, as commonly, replanning of motion (e.g., obstacle avoidance) is designed on the path planning level and motion is designed and constrained separately by motor controllers.

Traditional trajectory generation is commonly based on the assumption that initial and final conditions (e.g. velocity and acceleration for a 5th order polynomial) are equal to zero. The work of Ahn et al. [5] proposes a method, denoted arbitrary states polynomial-like trajectory (ASPOP), which designs a trajectory with arbitrary initial and final conditions. The method generates the trajectories online, however, constraints are not taken into account.

The work of Thompson et al. [6] describes a trajectory generation approach which explicitly considers the presence of obstacles. The method entails adding a fourth order term to a cubic polynomial and a cost function to the state equations. Solving for the parameters of the polynomial given initial and final conditions then generates polynomial trajectories which minimise the cost function.

The work of Namiki et al. [7] presents an online trajectory generator for catching a flying ball in mid-air. A 5th order polynomial is used to describe all possible target trajectories in the neighbourhood of the catching point. The parameters of the trajectory are optimized depending on the dynamics and the kinematics of the manipulator and the object. A final trajectory is then generated such that the end-effector can catch the target at one point, and a match between the position, velocity, and acceleration of the target and the end-effector is satisfied.

Motion planning in visual servoing can be found in [8]. In this, visibility and workspace constraints are considered while minimizing a cost function (e.g., spanned image area, trajectory length). As trajectories are designed in image space, only a (image) path is designed.

A complete framework for generating trajectories online is presented in the work of Kröger [9], [10]. Particularly motion systems subject to unforeseen events benefit from this approach by being able to directly react to events and...
switch between different control methods or domains. As such, this is a hybrid switched systems approach to robotic manipulation and is motivated to generate motion with arbitrary initial conditions. Experiments present a trajectory generation in which the final conditions can be specified up to and including velocity (i.e., 3\textsuperscript{rd} order) and the acceleration is set to zero.

Our approach does not consider switching between different control methods but considers changes of the trajectory itself. In particular, this entails changes due to obstacle avoidance and therefore adaptation of intermediate and final conditions (i.e., position, velocity and acceleration on via-point and end-point). This means that every iteration, a new trajectory is designed with different conditions. As such, visual measurements can be incorporated in a constrained manner.

The problem of reaching a given constraint is solved online by a computationally efficient optimization procedure that iterates to a required motion bound (e.g. velocity, acceleration) by extending the execution time. This guarantee effectively makes the generated trajectory time-optimal. Finally, synchronization between constraints as well as synchronization between DOFs is considered prior to motion generation.

The remainder of this paper is organized as follows. Section II recalls traditional trajectory planning with a minimum jerk polynomial. Section III discusses the proposed direct trajectory generation method and its use for obstacle avoidance. Finally, section IV presents simulation and experimental results.

II. TRADITIONAL MOTION GENERATION

Traditional motion generation can be divided in two categories: offline motion planning and sensor-based motion planning (e.g., visual servoing). Commonly, sensor-based motion planning designs a path, whereas offline motion planning designs a trajectory. The difference between path planning and trajectory planning is that a path only takes geometric considerations into account. A trajectory includes time and can therefore specify kinematic constraints. Following, both methods are explained and compared in more detail.

A. Vision-based vs. Offline Motion Planning

In offline motion planning, a trajectory is designed before any motion is executed. This trajectory cannot be changed at runtime, however, constraints on the trajectory can be easily considered. A common procedure is to execute multiple trajectories successively, where subsequent trajectories can account for changes in conditions or constraints. This implies that, while executing motion, the system is blind to any changes.

Sensor-based motion planning considers the motion of a system to be dependent on the sensor at hand, which means that the motion is directly modified based on the sensor readings. The design of this motion is usually highly simplified, as incorporation of sudden events is fairly complex or too time-consuming. In particular, in vision-based control, common procedure is to use an image error $e$ as feedback to control the velocity $v_m$ of a manipulator:

$$ v_m = L_q^\dagger \dot{e}, \quad \dot{e} = -\lambda e, \quad (1) $$

where $L_q^\dagger$ is the pseudo-inverse of the interaction matrix [11] and $\lambda > 0$ serves as a gain. This exponential error decrease can lead to non-smooth or undesirable robot motion. The initial error (step at $t = 0$) is discontinuous and kinematic constraints on the trajectory are not included. Furthermore, missing or delayed measurements have to be dealt with by e.g. an observer, otherwise instability of the system may occur.

An additional difference between both traditional methods is their execution time. Traditional offline motion planning defines a single control structure known as trajectory tracking, which can be executed at a high rate (e.g., 1 [kHz]). On the other hand, sensor-based control requires more processing time to compute a motion command (e.g., force or visual control). This gives rise to a local control loop that guarantees stability and a global loop that computes the path.

Closer inspection suggests that if both approaches could be adapted into one, the advantages of both could account for an improved motion design. This approach fits perfectly in a motion control scheme where direct reactions to sensor readings are eminent. More specifically, we propose an approach of direct trajectory generation as solution to the obstacle avoidance problem. Where path planning would direct an avoidance procedure merely on path planning level, our direct trajectory generation method considers the avoidance procedure on trajectory planning level and, as such, can consider motion constraints directly.

B. Minimum Jerk Trajectories

For simple trapezoidal trajectories, discontinuities occur during transition from constant to zero acceleration and during velocity reversal. This discontinuous acceleration will cause infinite jerk, which in turn tends to cause overshoot, electric noise in the power source and unwanted vibrations [12]. Furthermore, the larger the magnitude of the jerk is, the larger the variation of acceleration is. Smooth motion can therefore be obtained by choosing the trajectory as a $5^\text{th}$ order polynomial. This implies that its $6^\text{th}$ derivative is zero, which will minimize the integrated square jerk [13]:

$$ f_{J_{\text{min}}}(t) = \int_{t=0}^{T} \dot{q}^2(t) dt = \int_{t=0}^{T} \left[ \frac{d^3q(t)}{dt^3} \right]^2 dt. \quad (2) $$

This trajectory has the form:

$$ q(t) = a_1 + a_2 t + a_3 t^2 + a_4 t^3 + a_5 t^4 + a_6 t^5 \quad (3) $$

for $0 \leq t \leq T$. The velocity and acceleration can be written as

$$ \dot{q}(t) = a_2 + 2a_3 t + 3a_4 t^2 + 4a_5 t^3 + 5a_6 t^4, $$

$$ \ddot{q}(t) = 2a_3 + 6a_4 t + 12a_5 t^2 + 20a_6 t^3. \quad (4) $$

By setting up a system of linear equations a Vandermonde [4] matrix $T$ can be formulated. This effectively performs
a polynomial interpolation, since solving the system of linear equations \( Ta = q \) for \( a \) is equivalent to finding the coefficients of the polynomial.

A 5\(^{th}\) order polynomial implies that \( T \) becomes \( 6 \times 6 \). With the points \((t_k, q_k), k \in \{i, f\}\) and considering additional constraints regarding initial \((i)\) and final \((f)\) velocities and accelerations, we can build the vectors \( q, a, \) and matrix \( T \) of order \( n + m \) (i.e. \( n + 1 = 2 \) points, \( m = 4 \) additional constraints) as:

\[
q = \begin{bmatrix} q_i & q_f & v_i & \alpha_i & v_f & \alpha_f \end{bmatrix}^T = Ta = \begin{bmatrix} 1 & t_i & t_i^2 & t_i^3 & t_i^4 & t_i^5 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_i & 3t_i^2 & 4t_i^3 & 5t_i^4 \\ 0 & 0 & 2 & 6t_i & 12t_i^2 & 20t_i^3 \\ 0 & 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 \\ 0 & 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}.
\]

(5)

The coefficients can then be recovered with \( a = T^\dagger q \), where \( T^\dagger \) is the pseudo-inverse of \( T \). Aforementioned method generates a minimum jerk trajectory for one degree of freedom. For multiple degrees of freedom, an equal number of trajectories has to be generated.

C. Trajectory Synchronization

Incorporating kinematic constraints into basic motion profiles is fairly straightforward. The duration (i.e., execution time) of the trajectory scales linearly (velocity) or square-root linearly (acceleration) with the affected constraint [4]:

\[
t_{s,l} = \left\{ \begin{array}{l} \frac{15}{8} \frac{h}{v_{max}} \sqrt{\frac{10\sqrt{3}}{3}} \frac{h}{\alpha_{max}} \end{array} \right\}
\]

(6)

where \( t_{s,l}, l \in \{v, \alpha\} \) is the execution time, \( h = q_f - q_i \) and \( v_{max} \) and \( \alpha_{max} \) are the maximum velocity and acceleration respectively. The constraint that is most critical is then chosen as the execution time. Considering multi-dimensional trajectories, it is highly unlikely that all motion is finalized at the same time instant. The standard approach is then to evaluate all constraints over all degrees of freedom and select the time that is most critical.

III. DIRECT TRAJECTORY GENERATION FOR OBSTACLE AVOIDANCE

Incorporating obstacle avoidance into a trajectory planner implies that constraints can be directly taken into account. In order to show the flexibility of the proposed approach, several methods for avoidance are presented, namely, obstacle avoidance for point-to-point motion and multi-point motion.

A. Direct Trajectory Generation

Direct motion planning requires that the order of the trajectory \((C^2)\) and the global constraints have to be defined beforehand. An outline of the proposed trajectory generation method is shown in pseudo-code in Algorithm 1. When an obstacle blocks the original end-point, a new end-point position \(q_e \) is computed (from vision) for avoidance. The proposed algorithm allows for event-based or rate-based obstacle avoidance (see Algorithm 1: line 1). For event-based avoidance, a trajectory update is incorporated only when an event occurs. In rate-based avoidance, the trajectory is updated continuously enabling even small disturbances to be incorporated. One downside on the latter approach is that noise can affect the generation quite significantly.

Algorithm 1 Direct Trajectory Generation (DTG)

Input: \( C^n, q_i, q_f \) \{initial conditions\}

\[
|S(k + 1)| \{next step state\}
\]

1. if \( q_s > 0 \) \( \| \mod (i, 10) = 0 \) then \{event or rate-based\}

2. compute \( t_{ev}, t_{ec} \) \{see algorithm 2\}

3. \( q_i = q(k - 1) \)

4. update \( q, T, t_f \)

5. \( q_f = q_s \)

6. \( v_i = v(k - 1) \)

7. \( \alpha_i = \alpha(k - 1) \)

8. \( t_f = t_s + t_e - \Delta t \)

9. \( S(k + 1) = [q_{k+1}, \dot{q}_{k+1}, \ddot{q}_{k+1}]^T \) \{see (3) and (4)\}

B. Obstacle Detection

For obstacle detection the 'SURF' [14] feature detector is employed. Greyscale images of planar obstacles are preloaded in memory of the pc and searched for continuously. Subsequent processing involves a homography estimation and decomposition [15] to obtain a Cartesian position (rotation is not considered for avoidance). As only a scaled translation can be recovered from the homography decomposition, a large margin is taken to avoid the obstacle.

C. Point-to-Point vs. Multi-point

When designing a trajectory with two points, the final point is the only variable that can be altered to avoid obstacles. This implies that after the trajectory moves away from an obstacle, still a new goal position has to be employed to move to a final end-goal.

If the trajectory is designed with multiple points, more design choices become available. For instance, via-points can be used to manœuvre around (multiple) obstacles where each via-point accounts for one obstacle. The final point then does not necessarily need to be adapted as via-points take care of avoidance. Furthermore, the constraints on the via-point(s) can be limited to only position, as a continuous \( C^2 \) trajectory is already guaranteed. Moreover, this choice is preferable, since in higher order trajectories, the behaviour becomes more oscillatory (i.e., Runge's phenomenon). With the addition of \( n \) via-points, the degree of the trajectory will grow with either \( 1n, 2n \) or \( 3n \) degrees depending on the number of constraints. Unfortunately, if a via-point increases the order of the polynomial trajectory, a minimum-jerk trajectory is no longer guaranteed.

D. Constraint Optimization

When considering that a new trajectory can be generated at any arbitrary state, a relation between time and
constraints is difficult to obtain. A straightforward solution is to optimize these constraints online. This implies that, every iteration, the constraints are evaluated and if, due to a redesign of the trajectory, these would be violated, additional time is added to the trajectory. The location of a current constraint (maximum or minimum) is found by computing the zero-crossings of the derivative (roots) of the considered polynomial and its magnitude by evaluating the original polynomial at these roots. A steepest descent optimization routine [16] is sufficient to accommodate for an eventual constraint mismatch and does not need to be executed in one iteration. For a velocity and acceleration constraint this is respectively expressed as
\[ t_{ev} = d_v(|v_m| - v_{max}), \]
\[ t_{ea} = d_a(|\alpha_m| - \alpha_{max}), \]
in which \( d_v > 0 \) and \( d_a > 0 \) define the rate of convergence, \( v_{max} \) and \( \alpha_{max} \) denote the predefined constraints, and \( v_m \) and \( \alpha_m \) the computed constraint (maximum or minimum) of the current trajectory.

As the optimization converges quite rapidly, the computations can be spread out over several iterations. Algorithm 2 presents more details of the optimization procedure in pseudo-code for point-to-point motion. For multi-point trajectories, the root solving problem becomes higher order, however, the solution method remains the same.

**Algorithm 2 Constraint Optimization**

**Input:** \( T(a), v_{max}, \alpha_{max} \) (trajectory and constraints)

**Output:** \( t_{ev} \parallel t_{ea} \) (extra time to satisfy constraint)

1. \( T_{vc} = 2a_3 + 6a_4t + 12a_5t^2 + 20a_6t^3 \) \{velocity constraint\}
2. \( T_{ac} = 6a_4 + 24a_5t + 60a_6t^2 \) \{acceleration constraint\}
3. if \( T_{vc} \) then
   4. \( T_{vc} = 0 \) \{find roots and sort descending in \( r \)\}
   5. \( t_m = r(2) \) \{time of maximum\}
   6. \( v_m = a_2 + 2a_3t_m + 3a_4t_m^2 + 4a_5t_m^3 + 5a_6t_m^4 \)
   7. if \( v_m > v_{max} \) then
      8. \( t_{ev} = d_v(|v_m| - v_{max}) \) \{steepest descent\}
9. end if
10. end if
11. if \( T_{ac} \) then
12. \( T_{ac} = 0 \) \{find roots and sort descending in \( r \)\}
13. \( t_m = \arg \max \{T_{ac} = 0\} \) \{time of maximum\}
14. \( \alpha_m = 2a_3 + 6a_4t_m + 12a_5t_m^2 + 20a_6t_m^3 \)
15. if \( \alpha_m > \alpha_{max} \) then
16. \( t_{ea} = d_a(|\alpha_m| - \alpha_{max}) \) \{steepest descent\}
17. end if
18. end if

The additional time needed to avoid violating a constraint is added to the original trajectory time. The fact that every iteration a new trajectory is generated implies that the trajectory time is continuously decreasing (except when \( t_{ev} \) or \( t_{ea} \) are added) and is equal to zero at the end of the trajectory. More specifically, at \( t = 0 \) and \( t = t_f \) it holds that
\[ t(0) = t_f, \quad \text{and} \quad t(t_f) = 0. \]

When computing the trajectory online, the initial and final time is defined as
\[ t_i = 0, \quad \text{and} \quad t_f = t_s + t_e - \Delta t \] (9)
where \( t_s \) is obtained from (6) and \( t_e \) is obtained from (7). \( \Delta t \) is the ascending trajectory time approximated as \( \Delta t = T_i j \), with \( T_i \) the local loop time with iteration count \( j \).

For a multi-dimensional trajectory, synchronization entails that the final time \( t_f \) of all trajectories is equal. The via-point time is obtained similarly to the final time defined in (9). When a trajectory is altered during runtime and additional time \( t_e \) is added, this is passed on to all other trajectories. A possible constraint violation due to this addition is dealt with by the constraint optimization procedure.

Typical for vision-based control systems is the separation of a visual loop with cycle time \( T_v \) and a local control loop with cycle time \( T_l \), where \( T_l > T_v \). As every cycle a new trajectory is generated, the real-time requirement of the visual loop is now no longer necessary.

**IV. EXPERIMENTAL RESULTS**

In order to show that the method can generate from an arbitrary state the desired motion profiles as explained in section III, first results are shown for a single DOF. Following, results are presented with a 7-DOF redundant manipulator, where motion is designed in Cartesian space.

**A. Experimental Results for a Single DOF**

Direct trajectory generation for a single DOF is shown in Fig. 1, for constraint optimization of acceleration. The end-point of the trajectory is changed at \( t = 1.2 \) [sec]. It can be seen that the bound of \( |\alpha_{max}| = 1 \ [m/s^2] \) is not exceeded. Closer inspection of \( t_f \) shows that directly after the change in conditions, the execution time is increased to comply with the initially imposed bounds. In this case (for clarity), it is chosen to optimize to the new constraints in several iterations (one optimization step per iteration). However, due to limited number of steps necessary for convergence, it is also possible for the optimization to converge within one iteration.

![Direct Trajectory Generation](image)

Fig. 1. Direct trajectory generation with online end-point change. In order to comply with desired constraints \(|\alpha_{max}| = 1 \ [m/s^2]\), the end-time of the trajectory \( t_f \) is iteratively increased directly after \( t = 1.2 \) [sec]. Note that the acceleration profile is continuous.
B. Manipulator Kinematics: AMOR Arm

The selected robotic manipulator is the AMOR\(^1\) anthropomorphic arm from Exact Dynamics, B.V.\(^2\) (see Fig. 2, and [17]), which has 7-DOF and a gripper on its end-effector. All the matrices required for the implementation of the inverse kinematics algorithm are developed in C/C++. For visual processing, an industrial camera (Prosilica GE680M) takes greyscale images which are processed using the computer vision library OpenCV [18]. The output of the obstacle detection algorithm using SURF is shown in Fig. 3.

C. Results for Point-to-Point Whole-Arm Movements

In order to assess the obstacle avoidance method, a scenario is developed in which a robotic manipulator should execute a predefined planar positioning task, and is blocked by an obstacle at certain time and location. This means that from any arbitrary state, the manipulator should be guided to a new (online updated) end-point, while maintaining certain kinematic constraints. Fig. 4 and Fig. 5 show the simulation and experimental results for this scenario. New end-conditions are computed when the obstacle is detected and constraints are not violated.

D. Results for Multi-Point Whole-Arm Movements

A similar scenario is developed to generate the motion for obstacle avoidance with a multi-point trajectory containing 3 points (one via-point is added with only a position constraint, thus still ensuring \(C^2\) continuity). This allows controlling more parameters of the trajectory compared to point-to-point motion. The via-point and end-point are both determined to avoid the obstacle. Fig. 6 and Fig. 7 show the simulation and experimental results of this scenario. Once more, new end-conditions are computed when the obstacle is detected and constraints are not violated.

---

1 http://www.amorrobot.com  
2 http://www.exactdynamics.com
end-conditions are computed when the obstacle is detected and constraints are not violated. As the constraint is reached, time-optimality is guaranteed.

One issue that remains when designing a multi-point trajectory is the fact that, due to the addition of a via-point, the trajectory is now a $6^{th}$-degree polynomial, which no longer implies a minimum-jerk trajectory.

V. CONCLUSIONS

In this work, a method for direct and online trajectory generation is proposed in which constraints on via- and endpoint are taken into account at runtime. The method is suitable for point-to-point and multi-point trajectory generation, where arbitrary start- and end-states can be defined. In order to comply with predefined constraints, an optimization scheme ensures that the constraint is always reached when desired (but not violated), thus ensuring the time-optimality of the trajectory. Simulations and experiments on a 7-DOF manipulator show the effectiveness of the approach by avoiding obstacles smoothly and directly when detected.

REFERENCES