Weierstraß-Institut
für Angewandte Analysis und Stochastik
Leibniz-Institut im Forschungsverbund Berlin e. V.

Preprint ISSN 0946 – 8633

Task assignment, sequencing and path-planning in robotic welding cells

Chantal Landry[1], Wolfgang Welz[2], René Henrion[1], Dietmar Hömberg[1],

Martin Skutella[2]

submitted: August 1, 2013

[1] Weierstrass Institute
Mohrenstr. 39
10117 Berlin
Germany
E-Mail: chantal.landry@wias-berlin.de
rene.henrion@wias-berlin.de
dietmar.hoemberg@wias-berlin.de

[2] Institute of Mathematics
Technische Universität Berlin
Straße des 17. Juni 136
10623 Berlin
Germany
E-Mail: skutella@math.tu-berlin.de
welz@math.tu-berlin.de

No. 1825
Berlin 2013

2010 *Mathematics Subject Classification.* 49N90, 65D18, 90C27, 90C35, 90C90.

Key words and phrases. Discrete optimization, vehicle routing problem, optimal control problem, collision detection, motion planning, cooperative robots.

This work has been supported by the DFG Research Center MATHEON – Mathematics for key technologies.

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Leibniz-Institut im Forschungsverbund Berlin e. V.
Mohrenstraße 39
10117 Berlin
Germany

Fax: +49 30 20372-303
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

ABSTRACT. A workcell composed of a workpiece and several welding robots is considered. We are interested in minimizing the makespan in the workcell. Hence, one needs i) to assign tasks between the robots, ii) to do the sequencing of the tasks for each robot and iii) to compute the fastest collision-free paths between the tasks. Up to now, task assignment and path-planning were always handled separately, the former being a typical Vehicle Routing Problem whereas the later is modelled using an optimal control problem. In this paper, we present a complete algorithm which combines discrete optimization techniques with collision detection and optimal control problems efficiently.

1. INTRODUCTION

Efficient production lines are essential to ensure the competitiveness of car manufacturers. These lines are divided into workcells which are composed of a workpiece, several robots and some obstacles. Typical obstacles are the conveyor belt which connects the cells to each other. In a cell, the robots must perform several tasks on the workpiece without colliding with each other and with the obstacles. For instance, four robots have to make 50 weld points on a car door.

To have efficient production lines, a manufacturer must minimize the time taken to complete all the tasks in a workcell, i.e. the *makespan* of the workcell. For that purpose, one needs to assign tasks between the robots, to decide the sequencing of the tasks of each robot and to find collision-free trajectories between task locations for each robot. Up to now, the task assignment, the sequencing and the path-planning were computed by hand. The goal of this paper is to present an algorithm which minimizes the makespan in a cell composed of welding robots. We call this problem the Welding Cell Problem (WCP).

The task assignment, the sequencing and the path-planning can not be handled separately. On one hand, the task assignment and the sequencing depend on the traversal time between two tasks. This time is obtained by computing the fastest collision-free trajectory between these two given tasks. On the other hand, one needs to know between which tasks the robot motion-planning must be computed.

This paper is divided as follows. The Welding Cell Problem is presented in Section 2. Section 3 contains the algorithm which solves (WCP). This algorithm is an iterative method which couples discrete optimization with collision detection and optimal control problems. Details on path-planning are given in Section 4, whereas the collision detection is presented in Section 5.

2. THE WELDING CELL PROBLEM

Let us consider a workcell composed of m tasks located at the points P_i, $i = 1, \dots, m$, and K robots, denoted by $R_1, \dots, R_K$. Each robot has its own characteristics (size, weight, welding tongue, ...). Consequently, a robot may not be able to perform all tasks.

For simplicity, let us assume that the robots are convex polyhedra whose center of gravity is initially located at the points S_k, $k = 1, \dots, K$. Let V be a set containing the initial position of the robots, S_k, $k = 1, \dots, K$, and the task locations, P_i, $i = 1, \dots, m$. An arc of V is defined as a pair of points of the form (V_j, V_ℓ), where $V_j, V_\ell \in V$. With each robot R_k, a set of arcs A_k of V is associated. An arc (V_j, V_ℓ) belongs to A_k if and only if the robot R_k can move from V_j to V_ℓ. Since a task location V_j may not be reached by some robots, a set of arcs must be defined for each robot.

With each arc (V_j, V_ℓ) in A_k, a weight $t^k_{j,\ell}$ is associated. This weight is the time used by robot R_k to travel from V_j to V_ℓ. In graph theory, the tuple $G = (V, A_1, \dots, A_K)$ is called a *directed weighted graph*, see e.g. [9]. Therefore, (WCP) can be mathematically represented as a graph.

We associate with the graph G a Boolean function c which indicates if a collision occurs between two moving robots. The function c is defined as follows

$$c: \{1,\dots,K\} \times V \times V \times \{1,\dots,K\} \times V \times V \to \{0,1\}$$
$$(k, V_j, V_\ell, k', V'_j, V'_\ell) \mapsto \begin{cases} 0, & \text{if no collision,} \\ 1, & \text{otherwise.} \end{cases}$$

Hereabove, "no collision" means that no collision occurs between robot R_k moving from V_j to V_ℓ and robot $R_{k'}$ moving from V'_j to V'_ℓ at the same time.

Let us define now the *tour*, T^k, of robot R_k. The tour T^k is a sequence of nodes in V whose first and last nodes are the initial position of the robot. More precisely, if n_k is the number of nodes in T^k and V_j^k denotes the j^{th} node in the sequence, then the tour T^k is the sequence $T^k = (V_j^k)_{j=1}^{n_k}$ that satisfies

1 $V_1^k = V_{n_k}^k = S_k$,
2 $V_j^k \in \{P_1,\dots,P_m\}$ for $j \in \{2,\dots,n_k-1\}$,
3 $(V_j^k, V_{j+1}^k) \in A_k$, for $j = 1,\dots,n_k-1$.

With each tour, we associate the total traversal time, τ^k, which is defined as the sum of the traversal time of each arc:

$$\tau^k = \sum_{j=1}^{n_k-1} t_{j,j+1}^k.$$

Two tours T^k and $T^{k'}$ are said *collision-free* if and only if $c(k, V_j^k, V_{j+1}^k, k', V_\ell^{k'}, V_{\ell+1}^{k'}) = 0$ holds for all pairs of arcs (V_j^k, V_{j+1}^k) and $(V_\ell^{k'}, V_{\ell+1}^{k'})$, $j = 1,\dots,n_k-1$ and $\ell = 1,\dots,n_{k'}-1$, such that the arcs are travelled by the robots at the same time.

Eventually, let L denote an upper bound of the time needed to complete the job in the welding cell. On the basis of these definitions, the Welding Cell Problem (WCP) can be formulated as follows:
(WCP): *Find tours T^k, $k = 1,\dots,K$, and their corresponding traversal times $t_{j,j+1}^k$, $j \in \{1,\dots,n_k-1\}$, such that:*

1 *every task is visited once:*
$$\cap_{k=1}^K T^k = \emptyset \text{ and } \cup_{k=1}^K T^k = V;$$
2 *the tours are pairwise collision-free: for all $k, k' \in \{1,\dots,K\}$, $k \neq k'$, we have*
$$c(k, V_j^k, V_{j+1}^k, k', V_\ell^{k'}, V_{\ell+1}^{k'}) = 0,$$
$\forall j \in \{1,\dots,n_k-1\}, \forall \ell \in \{1,\dots,n_{k'}-1\}$ *such that the arcs (V_j^k, V_{j+1}^k) and $(V_\ell^{k'}, V_{\ell+1}^{k'})$ are travelled simultaneously;*
3 *the traversal times $t_{j,j+1}^k$, $j \in \{1,\dots,n_k-1\}$ and $k = 1,\dots,K$, are minimized*
4 *the total traversal times τ^k, $k = 1,\dots,K$, are upper bounded by L:*
$$\tau^k = \sum_{j=1}^{n_k-1} t_{j,j+1}^k \leq L, \forall k \in \{1,\dots,K\}.$$

So far (WCP) is just a feasibility problem to find tours T^k, $k = 1,\dots,K$, with makespan less than L. This can, however, be easily extended to an optimization problem by giving every tour a cost and then finding a solution where the sum of the tour costs is minimized. Thus, the tours T^k, which are solutions of (WCP), are in the following called the *optimal tours* of (WCP). In a second step, binary search over L can be used to identify the minimal possible makespan.

On one hand, the tour T^k indicates that the tasks performed by robot R_k are located at V_j^k, $j = 1, \ldots, n_k - 1$. On the other hand, the sequence of points in T^k gives the sequencing: robot R_k starts from its initial position S_k and moves to V_1^k. Once the task is performed at V_1^k, the robot moves to V_2^k and so on until the robot reaches the last task location $V_{n_k-1}^k$. After performing the last task, the robot goes back to its initial position S_k.

The path-planning is obtained when the minimized traversal times $t_{j,j+1}^k$, $j = 1, \ldots, n_k - 1$, are computed. Indeed, the path-planning of robot R_k between the points V_j^k and V_{j+1}^k is such that the path is travelled as fast as possible and without collision with the obstacles present in the workcell. Details on the computation of the path-planning are given in Section 4.

The second condition in (WCP) ensures that no collision occurs between the robots, whereas the collision avoidance between a robot and the obstacles is guaranteed when the traversal times $t_{j,j+1}^k$ are computed.

Eventually, note that the Boolean function c in (WCP) is a priori not known. At the beginning, no information on collisions between the robots exists. This function has to be determined during the resolution of (WCP). Details on the collision detection are given in Section 5.

The definition of (WCP) serves as a general basis for a whole set of problems. It is possible to add additional constraints to closer model the needs of the actual welding requirements, e.g. an order in which the points are processed or additional angle constraints for the welding tongues.

3. Resolution algorithm

To solve (WCP), one needs to know the value of all traversal times $t_{j,\ell}^k$, $(V_j, V_\ell) \in A_k$ and $k = 1, \ldots, K$. However, the traversal times are the solution of an optimal control problem (see Section 4) and their computation is time consuming. Consequently, we consider first approximated traversal times and compute the exact traversal times only as necessary.

The approximated times are the traversal times of approximated trajectories. The approximated trajectories are obtained by putting a regular grid on the workcell. The nodes of the grid which are located in or in the neighbourhood of an obstacle are removed from the set of nodes. The trajectory between two given points is the shortest path connecting these two points and defined on the modified grid. The shortest path is obtained by applying a Dijkstra-like algorithm in which the usage of nodes that are very close to the obstacles are penalized. In addition to that, we also add the constraint that the angles between two successive edges are minimized (see [9]). The resulting trajectory is a sequence of segment lines which do not collide with the obstacles.

An example of an approximated trajectory is given in Figure 1. This figure shows a two dimensional example which contains four obstacles (black quadrilaterals), five tasks located at P_i, $i = 1, \ldots, 5$, and 2 robots (black and grey squares). Robot R_1 is initially located at S_1 and robot R_2 at S_2. The exact trajectory connecting S_2 to P_4 is shown in black, whereas the corresponding approximated trajectory is in grey. The approximated trajectory is collision-free with obstacles, but no more the fastest one.

If the collision function c and the exact traversal times are known, then (WCP) becomes a variant of the Vehicle Routing Problem (VRP) [15]. This problem can then be solved using Column Generation techniques, see [14] for details. Because of this fact, we choose an iterative algorithm to solve (WCP). The resolution algorithm loops on the following three steps until the optimal tours are found

1 Find tours T^k, $k = 1, \ldots, K$.
2 Update the traversal times of the arcs associated with the tours T^k, $k = 1, \ldots, K$.
3 Update the collision function c.

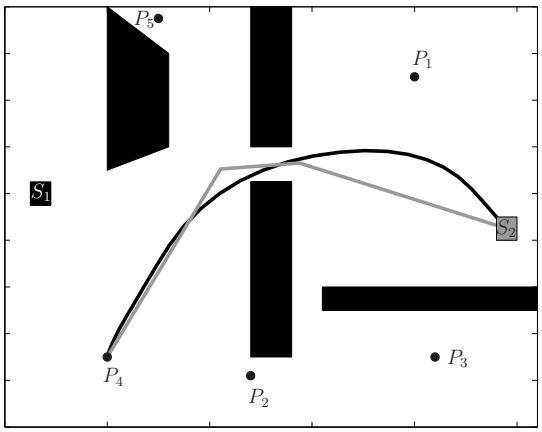

FIGURE 1. The black curve is the exact trajectory of robot R_2 which moves from S_2 to P_4. The corresponding approximated trajectory is the sequence of line segments illustrated in grey.

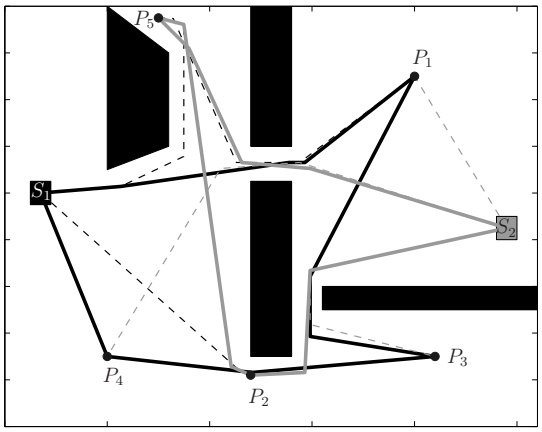

FIGURE 2. The tour T^1 based on the approximated trajectories is given by $(S_1, P_1, P_3, P_4, S_1)$ and represented by the black bold line. The tour of robot R_2 is $T^2 = (S_2, P_2, P_5, S_2)$ and depicted in grey. The dashed black, resp. grey, lines are the approximated trajectories related to the arcs in A_1, resp. A_2, which do not belong to the tour T^1, resp. T^2.

Let us apply this algorithm on the example depicted in Figure 1. At the beginning, all traversal times are initialized with the approximated trajectories and the c function is identical to the null function.

By applying the resolution technique in [14], the tours T^k are obtained for each robot R_k, $k = 1, \ldots, K$. Note that these tours are obtained whereas the weights on the arcs are the approximated traversal times. In Figure 2, the tours T^1 and T^2 based on the approximated trajectories are given for the robots R_1 and R_2.

In a second time, the exact paths and the associated traversal times between the points V_j^k and V_{j+1}^k, $j = 0, \ldots, n_k - 1$ belonging to the tour T^k, $k = 1, \ldots, K$, are computed by solving the optimal control problem presented in Section 4. At this step of the resolution, the arcs defined by the tours T^k have the exact traversal times as weights. The exact trajectories for the tour T^1 and T^2 are illustrated in Figure 3. In this figure, the arcs which do not belong to the tours are still associated with the approximated trajectories (dashed segment lines).

The tour T^k is such that no collision occurs between robot R_k and the obstacles. One needs now to check if the tours are collision-free. For that purpose, we consider the robots pairwise and look if a collision occurs between the robots. If so, we report the collision detection in the c function and look for new tours.

In the computation of the new tours, the exact traversal times are kept and we compute the exact trajectories only between the pair of points which were not considered in the tours up to now. For instance, if the new tour T^1 is the sequence (S_1, P_1, P_4, S_1), then only the trajectory between P_1 and P_4 is computed. The exact trajectories between S_1 and P_1, and between P_4 and S_1 were already computed.

We loop on the steps 1), 2) and 3) until the tours are pairwise collision-free. Eventually, one needs to check if these collision-free tours are still optimal. Indeed, if the tours were computed first with some approximated traversal times, then the tours with only exact traversal times may be not optimal since we changed the values of some weights. Therefore, new tours are computed based on the exact traversal times. If the new tours are identical to the previous one, we found the optimal solution.

The resolution algorithm can be summarized as follows

(i) Compute the approximated trajectories and the associated traversal time for all arcs in A_k, $k = 1, \ldots, K$.
(ii) Find the optimal tours T_k, $k = 1, \ldots, K$, based on the graph $G = (V, A_1, \ldots, A_K)$ and the c function.
(iii) Compute the exact trajectories for all arcs in T_k, $k = 1, \ldots, K$, such that the exact trajectory for these arcs is not computed yet.
(iv) If collisions between the tours are detected then
- Adapt the c function
- Go to (ii)

Else
- Find the optimal tours $\bar{T}_k$, $k = 1, \ldots, K$, based on the graph $(V, A_1, \ldots, A_K)$ and the c function
- If $\bar{T}_k$ is identical to T_k for all $k = 1, \ldots, K$, then
  return, *the optimal tours are found!*
  Else
  Go to (iii)
  Endif

Endif

The basic idea behind this approach is that in every iteration a relaxation of the original problem is solved and then more of the original information is added. Since our problem contains only finitely many arcs this algorithm will terminate after a finite number of steps. For a valid relaxation we have to make sure that the approximated trajectories are always an underestimation of the actual traversal time. Also, to guarantee that our algorithm finds the optimal solution of the original problem, we have to ensure that an infeasible sub problem in step (ii) implies the infeasibility of the master problem and that no feasible solution of the master is lost. The latter is only true, if we extend our (WCP) definition a little bit and also allow waiting in our tours, where the robot stays in the current node for some period in time. Fortunately, this idea can be easily incorporated in our approach without changing any of the described algorithms.

With this algorithm, only 19 exact trajectories among all the arcs in the graph G are computed for the two dimensional examples of Figures 1-3. In the next sections, details on the steps (iii) and (iv) of the algorithm are given.

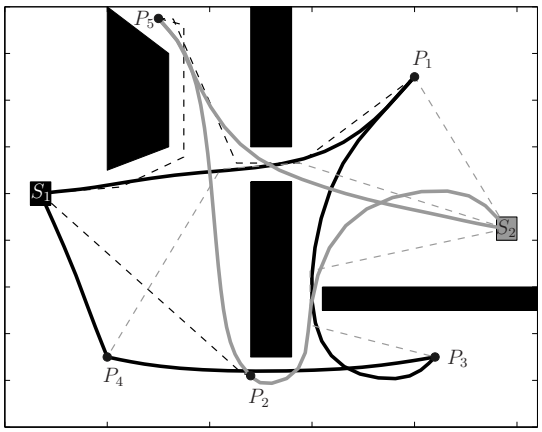

FIGURE 3. The tours $T^1 = (S_1, P_1, P_3, P_4, S_1)$ and $T^2 = (S_2, P_2, P_5, S_2)$ with exact trajectories. The dashed lines are approximated trajectories which were not considered in a tour yet.

4. PATH-PLANNING ALGORITHM

In this section, we present a method to compute the path-planning of a robot that moves between two given task locations.

Let us consider a robot composed of p links which are connected by revolute joints. Let $q = (q_1, \ldots, q_p)$ denote the vector of joint angles at the joints of the robot. Moreover, let the vector $v = (v_1, \ldots, v_m)$ contain the joint angle velocities and let $u = (u_1, \ldots, u_m)$ describe the torques applied at the center of gravity of each link. The robot is asked to move as fast as possible from a given position to a desire location. Its motion is given in the Lagrangian form as follows

$$\begin{aligned} q^{'}(t) &= v(t) \\ M(q(t))\, v^{'}(t) &= G(q(t), v(t)) + F(q(t), u(t)), \end{aligned} \tag{1}$$

where $M(q)$ is the symmetric and positive definite mass matrix, $G(q, v)$ contains the generalized Coriolis forces and $F(q, u)$ is the vector of applied joint torques and gravity forces The function F is linear in u.

The motion of the robot must follow (1), but also be collision-free with the obstacles present in the workcell. For simplicity, let us assume that only one obstacle is present. To establish the collision avoidance condition, the robot and the obstacle is approximated by a union of convex polyhedra. The approximation is denoted by P for the robot, by Q for the obstacle and are given by $P = \cup_{i=1}^{p} P_i$, with $P_i = \{x \in R^3 | A^{(i)} x \leq b^{(i)}\}$ and $Q = \cup_{j=1}^{q} Q_j$, with $Q_j = \{x \in R^3 | C^{(j)} x \leq d^{(j)}\}$, where $A^{(i)} \in R^{p_i \times 3}$, $b^{(i)} \in R^{p_i}$, $C^{(j)} \in R^{q_j \times 3}$, $d^{(j)} \in R^{q_j}$, and p_i and q_j are the number of faces in P_i and Q_j, respectively.

The robot P and the obstacle Q do not collide if and only if for each pair of polyhedra (P_i, Q_j), $i = 1, \ldots, p$, $j = 1, \ldots, q$, there exists a vector $w^{(i,j)} \in R^{p_i + q_j}$ such that: $w^{(i,j)} \geq 0$,

$$\begin{pmatrix} A^{(i)} \\ C^{(j)} \end{pmatrix}^T w^{(i,j)} = 0 \text{ and } \begin{pmatrix} b^{(i)} \\ d^{(j)} \end{pmatrix}^T w^{(i,j)} < 0. \tag{2}$$

This is a direct consequence of Farkas's lemma. See [6] for more details.

The fastest trajectory of a robot is the solution of an optimal control problem where the system of ordinary differential equations (ODE) are given by (1), see [3]. If an obstacle is present in the workcell, the collision avoidance is guaranteed as soon as the vector $w^{(i,j)}$ of (2) is found at each time t and

for all pairs of polyhedra. However, to be written as state constraints, the strict inequality in (2) has to be relaxed. Furthermore, since the robot moves, the matrices $A^{(i)}$ and the vectors $b^{(i)}$ evolve in time. Their evolution depends explicitly on $q(t)$. A complete formulation of $A^{(i)}(q(t))$ and $b^{(i)}(q(t))$ is given in [6].

Finally, the optimal control problem to find the fastest collision-free trajectory is given by:

(OCP): *Find the traversal time* t_f*, the state variables* $q,\ v\ :\ [0,t_f]\ \rightarrow\ R^p$*, and the controls* $u\ :\ [0,t_f]\ \rightarrow\ R^p$ *and* $w^{(i,j)}\ :\ [0,t_f]\ \rightarrow\ R^{p_i+q_j}$*,* $i\ =\ 1,\ldots,p,\ j\ =\ 1,\ldots,q$ *such that* t_f *is minimized subject to*

i) *the ordinary differential equations*

$$\begin{aligned} q^{'}(t) &= v(t) \\ v^{'}(t) &= M(q(t))^{-1}\left(G(q(t),v(t))+F(q(t),u(t))\right); \end{aligned}$$

ii) *the state constraints for* $i=1,\ldots,p,\ j=1,\ldots,q$

$$\begin{aligned} \begin{pmatrix} A^{(i)}(q(t)) \\ C^{(j)} \end{pmatrix}^T w^{(i,j)}(t) &= 0 \\ \begin{pmatrix} b^{(i)}(q(t)) \\ d^{(j)} \end{pmatrix}^T w^{(i,j)}(t) &\leq -\varepsilon, \end{aligned} \tag{3}$$

iii) *the boundary conditions* $R(q(0))-V_0=0$*,* $v(0)=0$*,* $R(q(t_f))-V_f=0$ *and* $v(t_f)=0$*; and*
iv) *the box constraints* $u_{\min}\ \leq\ u\ \leq\ u_{\max}$ *and* $0\ \leq\ w^{(i,j)}$*,* $i=1,\ldots,p,\ j=1,\ldots,q$*,*

where $R(q)$ denotes the position of the barycentre of the last link of the robot and V_0, V_f are the given task locations. The vectors u_{min} and u_{max} are also given and the relaxation parameter ε is positive and small.

(OCP) can be easily applied with several obstacles. It suffices to define control variables and to write (3) for each obstacle. Depending on the number of state constraints (3), the problem is inherently sparse since the artificial control variables $w^{(i,j)}$ do not enter the dynamics, the boundary conditions, and the objective function of the problem, but only appear linearly in (3).

The optimal control problem (OCP) is solved with the package OC-ODE developed by Matthias Gerdts [5]. The method involves first discretizing the control problem and transforming it into a finite-dimensional nonlinear optimization problem. The control variables are approximated by B-splines of order 2 and the ordinary differential equations are integrated with the classical Runge-Kutta method of order 4. The resulting nonlinear optimization problem is then solved by a sequential quadratic programming method [4, 7]. As in [12] we use an Armijo type line-search procedure for the augmented Lagrangian function in our implementation. However, the resulting optimization problem contains a lot of constraints: at each time step of the control grid and for all pairs of polyhedra $(P^{(i)}, Q^{(j)})$, four state constraints are defined (compare (3)). To reduce the number of constraints and variables, we add an active set strategy based on the following observation: the state constraints are superfluous when the robot is far from the obstacle or moves in the opposite direction. The active set strategy is fully detailed in [6].

5. Collision detection

This section concerns the detection of collisions between two robots moving along a given path. There exist two types of collision detection. The *static* collision detection checks if there is a collision between two objects at each time step. The *dynamic* collision checking determines if for all configurations given on a continuous path a collision occurs between the objects. As Cameron pointed out in [1], the static detection is simple, but can miss a collision if the time discretization is too rough. On the other side, taking small time steps is time consuming. For this reason, we choose to use a dynamic collision

method. We follow the method developed by Schwarzer, Saha and Latombe in [13]. This method is based on the comparison of lower bounds of the distance between the robots with an upper bound of the relative distance travelled by the points in the robots. The advantages of the method are its simplicity, its exactness and the automatic adaptation of the sampling resolution.

Let us consider two robots, R_1 and R_2. Let us assume for simplicity that the robots are a convex polyhedron. Each robot moves along a given path. This path was computed in (OCP). To solve (OCP), a time discretization was used. Each robot has its own time discretization. Let $(t_i^1)_{i=1}^{N_1}$, resp. $(t_i^2)_{i=1}^{N_2}$, denote the discretization of robot R_1, resp. R_2.

Let us classify the time steps $(t_i^1)_{i=1}^{N_1}$ and $(t_i^2)_{i=1}^{N_2}$ in an ascending order. Then the time is decomposed on subintervals $[t_l, t_u]$ of the form: $[t_i^1, t_{i+1}^1]$, $[t_i^1, t_{j+1}^2]$, $[t_j^2, t_{i+1}^1]$ or $[t_j^2, t_{j+1}^2]$. We check on each such subinterval if a collision between the robots occurs.

Let us consider the time subinterval $[t_l, t_u]$. The idea of Schwarzer, Saha and Latombe is to compare upper bounds of the distance travelled by the points of the robots during $[t_l, t_u]$ with a lower bound of the distance between both robots. Let us define the following quantities

- $\eta(t)$ is a non-trivial lower bound of the Euclidean distance between the robots at time t. The relation $\eta(t) \leq \delta$, δ small and positive, means that the robots are colliding.
- $\lambda_1(t_a, t_b)$ is an upper bound on the length of the curves traced by all points in robot R_1 between t_a and t_b with $t_a, t_b \in [t_l, t_u]$.
- $\lambda_2(t_a, t_b)$ is an upper bound on the length of the curves traced by all points in robot R_2 between t_a and t_b with $t_a, t_b \in [t_l, t_u]$.

Schwarzer, Saha and Latombe's method is based on the following sufficient condition:
Two polyhedra R_1 and R_2 do not collide at any time $t \in [t_l, t_u]$ if

$$\lambda_1(t_l, t_u) + \lambda_2(t_l, t_u) \ < \ \eta(t_l) + \eta(t_u). \tag{4}$$

The reverse of the above condition is not true. We cannot say anything about the collision-freeness on $[t_l, t_u]$ when the inequality is not satisfied. In that case, the idea is to bisect the time interval into two subintervals $[t_l, t_m]$ and $[t_m, t_u]$ where $t_m = \frac{1}{2}(t_l + t_u)$. In a second time, we check if a collision occurs at t_m by computing $\eta(t_m)$. If $\eta(t_m)$ is positive, then the sufficient condition is applied on both subintervals $[t_l, t_m]$ and $[t_m, t_u]$. So, the core algorithm to determine if there is a collision between R_1 and R_2 on $[t_l, t_u]$ is given by

Collision detection algorithm:
If $\eta(t_l) \ \leq \ \delta$ or $\eta(t_u) \ \leq \ \delta$ then
    return *collision*
else
    return Inequality(t_l, t_u)
end if

where Inequality (t_a, t_b) is a recursive function defined as

if $\lambda_1(t_a, t_b) + \lambda_2(t_a, t_b) \ < \ \eta(t_a) + \eta(t_b)$ then
    return *no collision*
else if $\eta(\frac{1}{2}(t_a + t_b)) \ \leq \ \delta$ then
    return *collision*
else
    return Inequality $(t_a, \frac{1}{2}(t_a + t_b))$ or

Inequality $(\frac{1}{2}(t_a + t_b), t_b)$
end if

The collision is detected once the lower bound η is smaller than a given threshold $\delta > 0$. Doing so, we ensure that the polyhedra remain at a safety distance from each other.

One strength of this detection method is that the algorithm automatically decides whether a time interval needs to be bisected further. Moreover, the method can never fail. To prove this fact, let us observe first that $\lambda_i(t_a, t_b) \rightarrow 0$ when $|t_b - t_a| \rightarrow 0$, $i = 1, 2$. Then, let us distinguish the cases:

- If no collision occurs in $[t_l, t_u]$, then there exists $\eta_{min} \geq \delta > 0$ such that $\eta(t) > \eta_{min}$, $\forall t \in [t_l, t_u]$. By bisecting, the length of the new time subintervals is always smaller. So, the left-hand side of Inequality (4) becomes smaller with the bisection whereas the right-hand side remains lower-bounded. Hence, Inequality (4) is satisfied.
- If the polyhedra collide, then there is a time subinterval $[t_a, t_b] \subseteq [t_l, t_u]$ such that $\eta(t) \leq \delta$, $\forall\, t \in [t_a, t_b]$ since the motion of the polyhedra is continuous. Then, by bisecting, Inequality (4) remains unsatisfied until the new middle of the time interval falls into $[t_a, t_b]$.

  Let us illustrate this argument with the example in Figure 4. The time interval $[t_l, t_u]$ is represented. The time interval $[t_a, t_b]$ when the collision occurs is in grey. The algorithm checks first if the polyhedra collide at t_l and t_u. Second step of the algorithm establishes that Inequality (4) is not satisfied. The first bisection is executed by computing $t_1 = \frac{1}{2}(t_l + t_u)$. No collision occurs at t_1 ($\eta(t_1) > \delta$). Inequality (4) is satisfied on $[t_l, t_1]$ but not on $[t_1, t_u]$. Hence, the middle point of $[t_1, t_u]$ is: $t_2 = \frac{1}{2}(t_1 + t_u)$. At that time, η is greater than δ. The bisection is then executed and we obtain the following subintervals $[t_1, t_2]$ and $[t_2, t_u]$. Inequality is verified on $[t_2, t_u]$ but not on $[t_1, t_2]$. Next, let us compute $t_3 = \frac{1}{2}(t_1 + t_2)$ and check if $\eta(t_3)$ is bigger than δ. Let us do so on until we reach $t_4 = \frac{1}{2}(t_3 + t_2)$. For that point, $\eta(t_4)$ is smaller than δ. The collision is detected.

Schwarzer, Saha and Latombe establish in [13] an upper bound λ_1, λ_2 for any kind of robots. The function η is defined as a non-trivial lower bound of the real distance between two polyhedra. A two-phase approach is considered which consists of a *broad phase* and *a narrow phase*. In the broad phase, the polyhedra are approximated by a simple bounding volume such as an axis-aligned box or a sphere and η is defined as the distance between the bounding volumes. As long as the bounding volumes are disjoint, the broad phase is applied. Once the bounding volumes overlap, the narrow phase is used. This phase computes the exact distance between the polyhedra. Thus the two-phase approach allows a minimal cost in the computation of η since the exact distance is determined only when the polyhedra are close to each other. Note that if the robots would have a more complex geometry, then a hierarchy of bounding volumes would be defined such as in [2, 8].

For the narrow phase, we use Lin and Canny's algorithm [10, 11]. This algorithm determines the closest pair of features between the polyhedra, where the features of a polyhedron are its vertices, its edges and its faces located on its boundary. We choose to follow Lin and Canny's algorithm since the approach is fast, easy to implement and perfectly suited when polyhedra move slightly between two time steps.

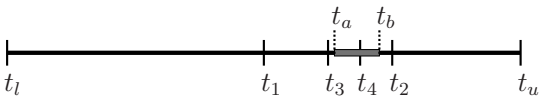

FIGURE 4. Convergence of the collision detection algorithm when a collision occurs.

6. Conclusion and perspectives

The Welding Cell Problem was presented. This problem involves minimizing the makespan of a workcell composed of several welding robots and tasks. The makespan is minimized when the tasks are optimally assigned between the robots, and the motion between the tasks is as fast as possible and collision-free. For that purpose, techniques from discrete optimization were efficiently combined with effective algorithms to solve optimal control problems and to detect collisions.

The formulation and the resolution of (WCP) are independent of the dimension of the workcell. In this article, we have presented numerical results in 2 dimensions. The application of (WCP) to a real welding cell is underway.

References

[1] S. Cameron. A study of the clash detection problem in robotics. In *Int. Conf. Robotics & Automation*, pages 488–493, 1985.

[2] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scaled environments. In *Symposium on Interactive 3D Graphics*, pages 189–196. ACM Siggraph, April 1995.

[3] M. Diehl, H. G. Bock, H. Diedam, and P. B. Wieber. *Fast Direct Multiple Shooting Algorithms for Optimal Robot Control*, pages 65–94. Springer, 2005.

[4] M. Gerdts. *Optimal Control of Ordinary Differential Equations and Differential-Algebraic Equations*. PhD thesis, Universität Bayreuth, 2006.

[5] M. Gerdts. *OC-ODE, Optimal Control of Ordinary-Differential Equations, Software User Manual*, April 2010.

[6] M. Gerdts, R. Henrion, D. Hömberg, and C. Landry. Path planning and collision avoidance for robots. *Numerical Algebra, Control and Optimization*, 2(3):437 – 463, 2012.

[7] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Rev.*, 47:99–131, 2005.

[8] S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In ACM SIGGRAPH, editor, *Computer Graphics Proceedings, Annual Conference Series*, 1996.

[9] R. Gould. *Graph theory*. Benjamin/Cummings Pub. Co., 1988.

[10] M. C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1993.

[11] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, page 1008, 1991.

[12] K. Schittkowski. On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function. *Mathematische Operationsforschung und Statistik*, 14:197–216, 1983.

[13] F. Schwarzer, M. Saha, and J. Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Tr. on Robotics*, 21:338–353, 2005.

[14] M. Skutella and W. Welz. Route planning for robot systems. In B. Hu, K. Morasch, St. Pickl, and M. Siegle, editors, *Operations Research Proceedings 2010*, pages 307–312. Springer, 2011.

[15] P. Toth and D. Vigo. The vehicle routing problem. pages 1–26. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.