Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 51-22 (2018) 220–225

Sensor-Based Task-Constrained Motion Planning using Model Predictive Control ⋆

Massimo Cefalo Emanuele Magrini Giuseppe Oriolo

Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza University of Rome, Italy
(e-mail: {cefalo,magrini,oriolo}@diag.uniroma1.it)

Abstract: A redundant robotic system must execute a task in a workspace populated by obstacles whose motion is unknown in advance. For this problem setting, we present a sensor-based planner that uses Model Predictive Control (MPC) to generate motion commands for the robot. We also propose a real-time implementation of the planner based on ACADO, an open source toolkit for solving general nonlinear MPC problems. The effectiveness of the proposed algorithm is shown through simulations and experiments carried out on a UR10 manipulator.

Keywords: Motion Planning, Obstacle Avoidance, Predictive Control

1. INTRODUCTION

As robots make their way into many application areas, collaborative robotics stands out as a rapidly expanding field that calls for specific advancements in planning and control techniques. The main issue is obviously that of reactivity: differently from conventional industrial robots that work in a complete information scenario, a collaborative robot must be able to work in a dynamic environment that is not a priori known and contains moving humans. Accordingly, the focus shifts from classical off-line motion planning to sensor-based techniques that allow the robot to quickly modify its motion in order to avoid collisions with obstacles detected by the sensory system.

Sensor-based motion planning methods can be roughly classified into two categories: *reactive* versus *replanning* approaches. Methods of the first category are devoid of any form of prediction or planning, even in the short term. They usually rely on kinematic control and react to obstacles through some form of artificial potential field; as a consequence, they are typically very simple to implement but may suffer from the appearance of local minima (Ogren et al., 2000; Haddadin et al., 2010, 2011; Flacco et al., 2012). Replanning algorithms are typically more complex and often combine a global motion planner with a local reactive controller (Ferguson and Stentz, 2006; Balan and Bone, 2006; Kuwata et al., 2009; Vannoy and Xiao, 2008; Belkouche, 2010; Chakravarthy and Ghose, 2012).

In this paper, we consider the sensor-based motion planning problem under the additional constraint that the robot system must execute a task. Task-constrained motion planning in complete information scenarios was considered by Oriolo and Vendittelli (2009) for static environments and by Cefalo and Oriolo (2014) for dynamic environments. To address the sensor-based version of the problem, we develop here an approach based on the use of nonlinear Model Predictive Control (MPC) to generate robot commands on the basis of the available measures. The task is expressed as a constraint whose satisfaction is embedded into the prediction model, while the cost function represents a trade-off between control effort and proximity of the robot to the obstacles. Depth camera images are processed by a parallel algorithm to compute robot-obstacle distances and detect imminent collisions.

MPC is an optimization-based method for controlling general systems by predicting their future behavior on the basis of dynamic models (Grune and Pannek, 2017). This approach is becoming increasingly popular in motion planning thanks to its capability to handle various kinds of constraints. Ide et al. (2011) use MPC for real-time trajectory planning of a mobile manipulator, and Frasch et al. (2013) for achieving obstacle avoidance of vehicles that move on low-friction road surfaces. Qian et al. (2016) propose an MPC-based methods for autonomous driving of urban vehicles while Bali and Richards (2017) use MPC in a robot navigation framework with obstacle avoidance.

Nonlinear MPC problems are, in general, solved by direct approaches: equations are discretized and the resulting structured nonlinear nonconvex problem is then solved. The algorithms used to solve such problems (e.g., interior point methods and sequential quadratic programming solvers) require many calculations and are therefore difficult to execute in real-time. To obtain a real-time algorithm, we generated the code for our MPC-based motion planner using ACADO, a software toolkit for automatic control and dynamic optimization which produces parallel C code suitable for real-time execution (Quirynen et al., 2015; Salaj et al., 2015). For the implementation, we resorted to parallel coding and GPU (Graphics Processing Unit) programming. GPU-based programming has known a strong diffusion in the last decade, also thanks to the availability of parallel computing platforms such as CUDA. This new programming approach is increasingly used also in robotics to obtain applications with real-time performance. In particular we exploited GPU capabilities to process depth images and compute robot-obstacle distances.

⋆ This work is supported by the EU H2020 project COMANOID.

10.1016/j.ifacol.2018.11.545

The paper is organized as follows. In Sect. 2 we give a formulation of the considered planning problem. Section 3 discusses the proposed approach from a general viewpoint. The sensor-based MPC planner, which is the core of the proposed method, is presented in Sect. 4. Section 5 discusses sensor-based distance computation. Section 6 describes simulation and experimental results, while future developments are mentioned in Sect. 7.

2. PROBLEM FORMULATION

Consider a robotic system whose configuration $\boldsymbol{q}$ (the vector of generalized coordinates) takes values in an n-dimensional configuration space $\mathcal{C}$. The system moves in a 3-dimensional workspace $\mathcal{W}$ populated by moving obstacles. Denote by $\mathcal{R}(\boldsymbol{q}) \subset \mathcal{W}$ the volume occupied by the robot at configuration $\boldsymbol{q}$, and by $\mathcal{O}(t) \subset \mathcal{W}$ the volume occupied by the obstacles at time t.

We will consider a *sensor-based* planning context, in that it is assumed that the obstacles' geometry and motion is not known in advance: there is a sensory system that provides a current view of the obstacles surrounding the robot. In addition, we assume that the obstacle move at sufficiently slow speed (in a sense that will be later explained) w.r.t. to rate of the MPC-based prediction phase. Based on this assumption, we can approximately consider the obstacles as stationary during the planning horizon, without the need to predict their motion.

The robot is assigned a task expressed in coordinates $\boldsymbol{y}$, which take values in an m-dimensional task space $\mathcal{Y}$. Typical task coordinates describe quantities related to manipulation, navigation or perception; in any case, they are related to the configuration coordinates by a forward kinematic map $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{q})$. In particular, the task is assigned as a desired *trajectory* $\boldsymbol{y}_d(t)$ for the $\boldsymbol{y}$ coordinates, with $t \in [0, t_f]$. Throughout the paper, we will assume that the considered robot is kinematically redundant with respect to the task, namely, $n > m$.

The considered motion planning problem consists in using the available sensory information in order to generate real-time motion commands for the robot so that

1. for all $t \in [0, t_f]$, it is $\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{q}(t)) = \boldsymbol{y}_d(t)$;
2. for all $t \in [0, t_f]$, it is $\mathcal{R}(\boldsymbol{q}(t)) \cap \mathcal{O}(t) = \emptyset$.
3. kinematic/dynamic robot capabilities are respected.

Condition 1 guarantees that the assigned task is continuously executed while condition 2 entails that all collisions with obstacles (including self-collisions) are avoided. The explicit form of condition 3 depends on the specific kind of robotic system under consideration. In particular, we focus on a velocity-controlled robotic manipulator, for which the command sent to the actuators is a reference generalized velocity $\dot{\boldsymbol{q}}$. Accordingly, we enforce position and velocity limits at the joint level (see Sect. 4.1).

Note that the real-time requirement is particularly challenging, since it requires that all computations related to planning are completed within the sampling interval T_c of the robot control module. In particular, this includes the distance computation, the generation of the control law and all collision checks necessary for its validation before sending it to the robot actuators for execution.

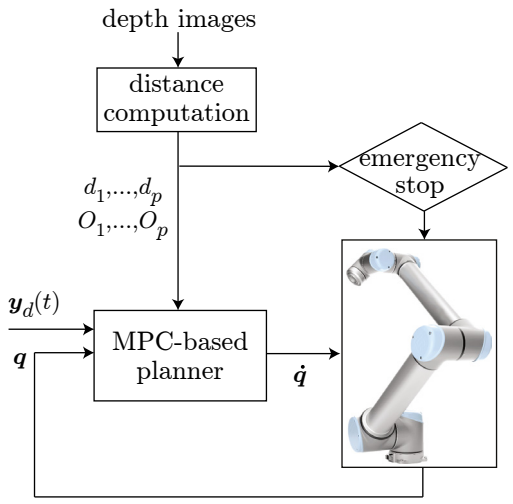

Fig. 1. Conceptual scheme of the proposed sensor-based motion planner.

3. THE PROPOSED APPROACH

To solve the problem of Sect. 2, we propose a sensor-based motion planner whose conceptual scheme is shown in Fig. 1. In this scheme we recognize three basic modules: distance computation, MPC-based planning and emergency stop.

For distance computation, we define a set of *control points* $P_1, \ldots, P_p$ which are appropriately distributed along the robot structure. By processing images from depth cameras, the distance computation module provides the following information for each control point ($i = 1, \ldots, p$) at the generic sampling time t:

- the point O_i of the obstacle region $\mathcal{O}(t)$ that is closest to P_i;
- the distance d_i between P_i and O_i.

Various sensory equipments can be used to implement the above functionality. In particular, in our implementation we have used a pair of depth cameras (see Sect. 5).

The MPC module is used to perform sensor-based planning on the basis of the current positions of the closest obstacle points $O_1, \ldots, O_p$. In particular, to guarantee the continuous execution of the task $\boldsymbol{y}_d(t)$, we use a constraint-aware prediction model for MPC computations. The cost function is a linear combination of two integral terms computed along the planning horizon: the first is aimed at keeping the robot away from the obstacles, while the second penalizes control effort. Constraints are added so as to guarantee the kinematic feasibility of the generated motion. As customary in MPC, only the first control action is actually sent out for execution, and a new optimization problem is set up and solved at the next time instant.

The distances $d_1, \ldots, d_p$ computed by the sensory system are used to perform an emergency stop procedure when their value goes below a certain threshold indicating an imminent risk of collision.

In the following, we describe in detail first the MPC-based planner, which is the core of our method, and then the sensor-based distance computation.

4. MPC-BASED PLANNER

The MPC planner uses the position of the obstacle points $O_1, \dots, O_p$ closest to the control points $P_1, \dots, P_p$ and the associated distances, provided by the distance computation module at each sampling instant.

4.1 MPC formulation

We adopt a constraint-aware MPC scheme to guarantee continuous task execution. In particular, the model is used for prediction is the discrete-time version of this continuous-time system:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\dot{\boldsymbol{y}}_d + \boldsymbol{K}\boldsymbol{e}_{\boldsymbol{y}}) + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\boldsymbol{w}, \tag{1}$$

where:

- $\boldsymbol{J}^{\dagger}$ is the pseudoinverse matrix of the task Jacobian $\boldsymbol{J} = \partial \boldsymbol{f}/\partial \boldsymbol{q}$;
- $\boldsymbol{K}$ is a positive definite gain matrix;
- $\boldsymbol{e}_{\boldsymbol{y}} = \boldsymbol{y}_d - \boldsymbol{y}$ is the task error;
- $\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J}$ is the orthogonal projection matrix in the null space of $\boldsymbol{J}$;
- $\boldsymbol{w}$ is the n-dimensional null-space velocity, which represents the available input.

At each time instant t_k, the MPC optimization problem is formulated as the constrained minimization of a cost function over a *planning horizon* $T_p = NT_c$, where N is a positive integer. The time interval involved in each problem is therefore $[t_k, t_{k+N}]$, with $t_{k+N} = t_k + NT_c$.

The constraints on the problem are expressed by the following inequalities

$$|\boldsymbol{q}| \leq \boldsymbol{q}^M \quad |\dot{\boldsymbol{q}}| \leq \dot{\boldsymbol{q}}^M, \quad t \in [t_k, t_{k+N}], \tag{2}$$

which must be interpreted as component-wise. Vectors $\boldsymbol{q}^M$ and $\dot{\boldsymbol{q}}^M$ represent respectively the bounds on generalized coordinates (joint limits) and their time derivatives (velocity limits).

The cost function is defined as

$$\min_{\boldsymbol{w}_k, \dots, \boldsymbol{w}_{k+N-1}} \sum_{j=k}^{k+N-1} \left(\sum_{i=1}^{p} \frac{1}{d_{i,j}^2} + \alpha |\boldsymbol{w}_j|^2 \right), \tag{3}$$

where $d_{i,j}$ is the *predicted* value of d_i (the distance between the i-th control point P_i and the closest obstacle point O_i) at the time instant $t_k + jT_c$ and α is a positive weight. As usual, only the first solution sample $\boldsymbol{w}_k$ is injected in (1) to generate the next robot velocity command.

To obtain the predicted distances $d_{i,j}$, $j = k, \dots, k+N-1$ that appear in (3), we predict the position of the i-th control point P_i at $t_k + jT_c$ via direct kinematics and compute its distance w.r.t. the closest obstacle point O_i at t_k as made available by the distance computation module.

4.2 Real-time implementation issues

For solving the MPC problem in real time, we generated an efficient parallel CPU code using ACADO (Quirynen et al., 2015), an open source toolkit that provides several integrators for solving nonlinear MPC problems. A special component of ACADO is the automatic code generation tool, which is based on real-time sequential quadratic programming solvers. Starting from a code that implements the definition of a MPC problem, this tool generates C++ code which is highly optimized for real-time applications. Quirynen et al. (2015) were able to solve practical relevant problems in a few microseconds.

In order to be able to use the ACADO automatic code generator and run the code in real time, we introduced some approximations in the problem described in Sect. 4.1. In particular, we approximated the nonlinear terms in the cost function as quadratic terms, using the first three elements of their Taylor series expansion at $\boldsymbol{q}_k$:

$$\frac{1}{d_i^{j\,2}} \simeq a + \boldsymbol{b}^T(\boldsymbol{q} - \boldsymbol{q}_k) + (\boldsymbol{q} - \boldsymbol{q}_k)^T \boldsymbol{Q}(\boldsymbol{q} - \boldsymbol{q}_k) \tag{4}$$

where

$$a = \frac{1}{d_i^{j\,2}}\bigg|_{\boldsymbol{q}_k} \quad \boldsymbol{b} = \left(\frac{\partial (1/d_i^{j\,2})}{\partial \boldsymbol{q}}\right)\bigg|_{\boldsymbol{q}_k} \quad \boldsymbol{Q} = \left(\frac{\partial^2 (1/d_i^{j\,2})}{\partial \boldsymbol{q}^2}\right)\bigg|_{\boldsymbol{q}_k}.$$

Equation (4) has been used to define a QP problem, neglecting a, which is a positive constant. Note that in order to have a well-defined QP problem, the matrix $\boldsymbol{Q}$ must be positive definite. To this end, we introduced a further approximation by replacing any negative eigenvalue of $\boldsymbol{Q}$ with a zero.

5. DISTANCE COMPUTATION

The distance computation module, which uses information in the form of depth camera images, is an extension of our previous technique (Cefalo et al., 2017).

Our method relies on the processing of three images, all with the same resolution:

- the *depth image*, which is a depth representation of the field of view of the camera;
- the *virtual robot image*, which contains a depth image of the robot from the same viewpoint reconstructed through direct kinematics from the CAD model and joint encoder readings at the current configuration;
- the *obstacle image*, obtained by subtracting the virtual robot image from the depth image.

The algorithm uses the depth information in the obstacle image to compute the distance between each robot control point and each obstacle point. In particular, we use the formula proposed by Flacco et al. (2012) for the distance between an obstacle point O and a robot point P:

$$d(O, P) = \sqrt{v_x^2 + v_y^2 + v_z^2}, \tag{5}$$

with

$$v_x = \frac{(o_x - c_x)d_O - (p_x - c_x)d_P}{f \cdot s_x}$$
$$v_y = \frac{(o_y - c_y)d_O - (p_y - c_y)d_P}{f \cdot s_y}$$
$$v_z = d_O - d_P,$$

where (o_x, o_y) and (p_x, p_y) are the x,y coordinates in depth space of O and P respectively, d_O and d_P are their depth, c_x and c_y are the pixel coordinates of the center of the image plane (on the focal axis), f is the focal length of the camera and s_x, s_y are the dimensions of a pixel in meters. The last five are the intrinsic parameters of the camera. Once all distances between the robot control point P_i ($i = 1, \dots, P$) and the obstacle points have been obtained,

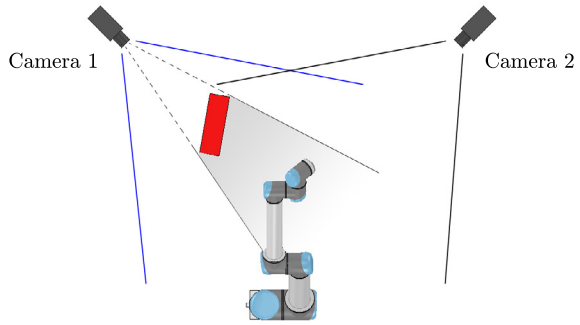

Fig. 2. A situation in which an obstacle (in red) occludes the robot for the first camera but not for the second.

the minimum value d_i is identified and the coordinates of the corresponding closest obstacle point O_i are stored.

To speed up the module, we process depth camera images and perform all computations directly on the GPU. Using the rendering library OpenGL and the shading language GLSL, the obstacle map is loaded into the GPU memory as a row vector with depth information. Distance computations with (5) are then performed in parallel for all pixels in the image using the CUDA framework.

So far, we have discussed distance computation using a single depth camera. However, this is an arrangement prone to errors due to the possibility of occlusions: if an obstacle is placed between the camera and the robot, it is impossible to evaluate the distances of the robot control points from the obstacle because its extension beyond the visible surface is unknown [1]. For example, in the situation of Fig. 2 one cannot compute robot-obstacle distances using only the depth image of Camera 1. From an algorithmic point of view, this is reflected in the fact that one or more robot control points are not visible in the depth image; in this case, the distance between these points and the obstacle cannot be computed using (5) and is set to zero, which is the worst-case estimate (*collision*).

To overcome the occlusion problem, we use two depth cameras connected to different GPUs that work in parallel on the same computer. Each GPU independently estimates robot-obstacle distances from its own depth image following the above procedure. Then, for each robot control point P_i only the minimum between the two estimates is retained, together with the associated obstacle point O_i.

If only one of the estimated distances is zero due to an occlusion, the other estimate is taken as the actual distance indicating that the robot is indeed in a collision-free configuration (see Fig. 2). If both estimated distances are zero, the algorithm returns a collision; this may still be a false positive due to both cameras being simultaneously occluded. However, our experimental results suggest that a careful placement of the cameras can significantly reduce the number of false positives.

6. RESULTS

The proposed sensor-based task-constrained motion planner was implemented in C++ and applied to a UR10 robot manipulator both in simulation and experiments (see `http://www.diag.uniroma1.it/labrob/research/MPC-TCMP.html` for video clips). For the planner, we set $\boldsymbol{K} = 5 \cdot \boldsymbol{I}$ in (1) and $\alpha = 0.5$ in (3); the planning horizon of the MPC is $T_p = 80$ ms. Joint and velocity limits in (2) were taken from the official documentation of the robot. The planner runs on a 64-bit Intel Core i7-4790 CPU at 4 GHz, equipped with a 16 GB DDR3 RAM and two high-performance graphic board NVIDIA GTX1080TI, each processing images from a different Kinect-2 depth camera. The distance computation module takes about 2.5 ms while the MPC controller provides velocity inputs in approximately 3 ms; this guarantees that the proposed motion planner actually runs in real time with a healthy margin with respect to the UR10 sample period $T_c = 8$ ms (i.e., 1/10 of the MPC planning horizon).

In all scenarios, the robot is assigned the purely orientational task of keeping its end-effector vertical. We use two control points, placed respectively at the elbow and wrist of the robot; these are sufficient for our purposes in view of the mechanical structure of the UR10. Around each control point we define a spherical safety area with a radius of 0.75 m: obstacle distances larger than this radius are set to zero in (3). If no obstacle enters the safety areas, only the norms of the $\boldsymbol{w}_j$'s appear in the cost function, whose minimization produces $\boldsymbol{w} = \boldsymbol{0}$ in (1) and thus the pure pseudoinverse solution. In other words, the sensor-based planner does not react to obstacles as long as they are outside the safety areas.

Figure 3 shows the result of a V-REP simulation where the UR10 must perform the assigned task in the presence of a human that crosses the scene passing in its vicinity. In the first two snapshots the robot is stationary because no obstacle is detected within the safety areas and therefore pure pseudoinverse control is applied. At $t = 3$ s, the first camera is partially occluded by the human and therefore detects a collision as worst-case estimate, while the second returns a nonzero distance indicating that there is actually no collision but the obstacle (the human's shoulder) has entered the elbow safety area. The sensor-based planner reacts to this situation by quickly moving the elbow away from the human ($t = 4$ s).

Figure 4 shows another simulation in the same exact scenario, but with the planner now using a sequential implementation of the distance computation module: in particular, the depth image is processed only in a small window (40×40 cm in Cartesian space) around each control point in order to respect bounds on the computation time due to the real-time requirement, similarly to what was done by Flacco et al. (2012). The robot is now able to detect the obstacle presence only at $t = 4$ s and therefore is much less reactive.

Finally, Fig. 5 illustrates an experiment on the real robot. The results confirm that our planner provides a remarkable level of reactivity: the UR10 continues to execute the assigned task but at the same time moves promptly away from the human, even when the latter intentionally tries to hit the robot wrist (snapshot 4) and elbow (snapshots 5 and 6). Moreover, as expected, the two-camera setup effectively overcomes the problem of occlusions.

[1] The same problem occurs when the robot is placed between the camera and the object.

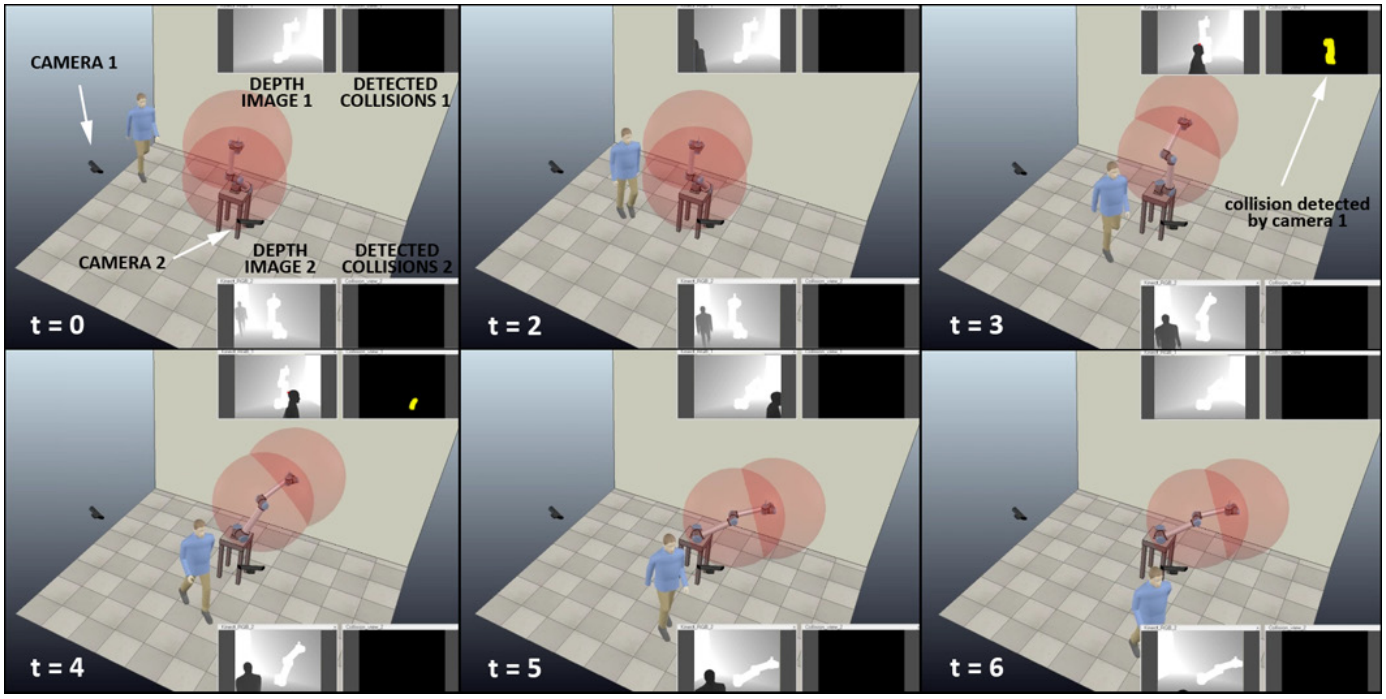

Fig. 3. Snapshots from a V-REP simulation. Each snapshot shows the depth image and detected collisions (worst-case estimates due to occlusions) for both cameras. Red spheres around the elbow and wrist control points represent the safety areas: the sensor-based planner reacts only to obstacles inside these areas.

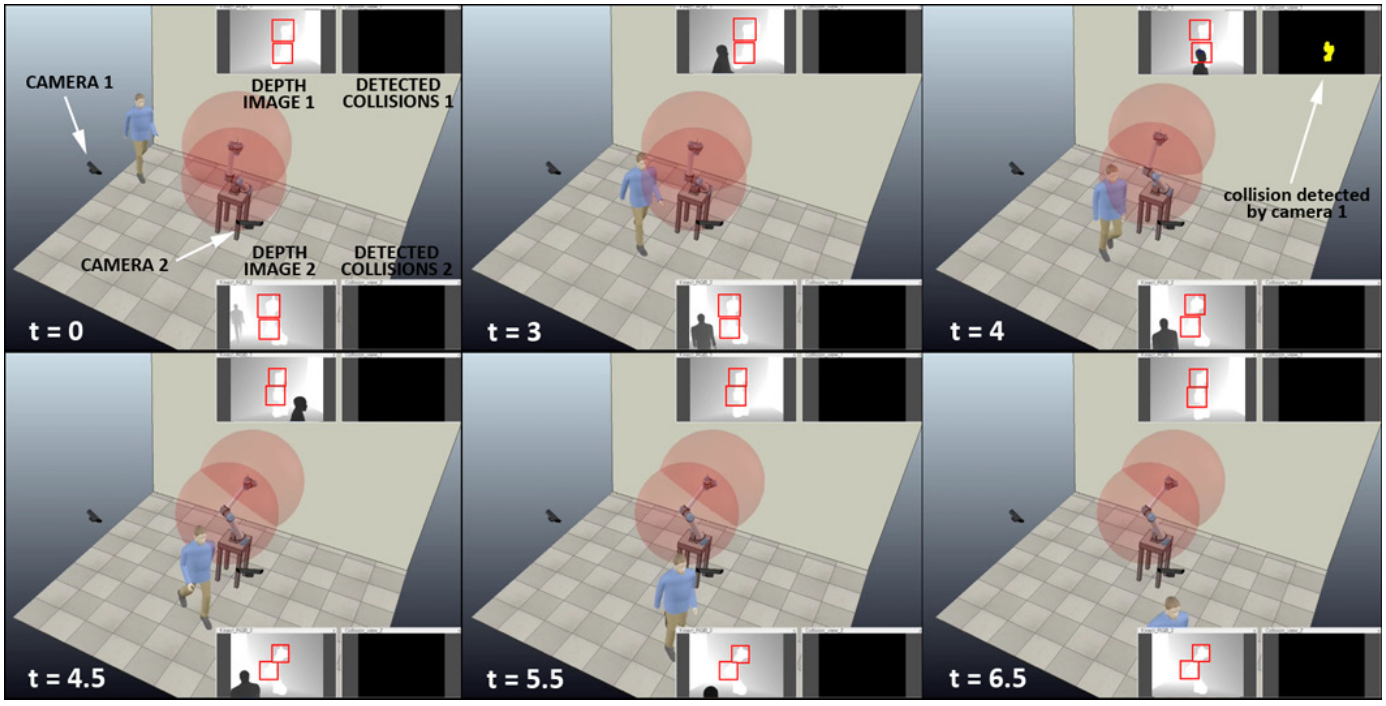

Fig. 4. Snapshots from another V-REP simulation in the same scenario, with the planner now using a sequential implementation of the distance computation module: depth images are only processed in the red squares around the control points.

7. CONCLUSIONS

We proposed a new sensor-based motion planning method for robots that must execute tasks in environments with moving obstacles. The method is based on nonlinear MPC for motion generation and incorporates an extended version of our previous distance computation algorithm now using images from a dual depth camera system. The effectiveness of the method was proven by simulation and experimental results on a UR10 manipulator.

Fig. 5. Snapshots from an experiment. Yellow and white arrows indicate, respectively, the motion direction of the obstacles and of the robot.

Future work will be aimed at improving the accuracy of robot-obstacle distance computations by setting up and exploiting a mapping between corresponding pixels of the two depth images. Another interesting possibility is predicting the motion of the obstacles by associating local displacements primitives to them.

REFERENCES

Balan, L. and Bone, G.M. (2006). Real-time 3D collision avoidance method for safe human and robot coexistence. In *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 276–282.

Bali, C. and Richards, A. (2017). Robot navigation using convex model predictive control and approximate operating region optimization. In *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2171–2176.

Belkouche, F. (2010). Reactive path planning in a dynamic environment. *IEEE Trans. on Robotics*, 25(4), 902–911.

Cefalo, M., Magrini, E., and Oriolo, G. (2017). Parallel collision check for sensor based real-time motion planning. In *2017 IEEE Int. Conf. on Robotics and Automation*, 1936–1943.

Cefalo, M. and Oriolo, G. (2014). Dynamically feasible task-constrained motion planning with moving obstacles. In *2014 IEEE Int. Conf. on Robotics and Automation*, 2045–2050.

Chakravarthy, A. and Ghose, D. (2012). Generalization of the collision cone approach for motion safety in 3D environments. *Autonomous Robots*, 32, 243–266.

Ferguson, D. and Stentz, A. (2006). Using interpolation to improve path planning: The field D* algorithm. *Journal of Field Robotics*, 23(2), 79–101.

Flacco, F., Kroger, T., and De Luca, A. (2012). A depth space approach to human-robot collision avoidance. In *2012 IEEE Int. Conf. on Robotics and Automation*, 338–345.

Frasch, J.V., Gray, A., Zanon, M., Ferreau, H.J., Sager, S., Borrelli, F., and Diehl, M. (2013). An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles. In *2013 European Control Conf.*, 4136–4141.

Grune, L. and Pannek, J. (2017). *Nonlinear Model Predictive Control – Theory and Algorithms*. Springer.

Haddadin, S., Belder, S., and Albu-Schaeffer, A. (2011). Dynamic motion planning for robots in partially unknown environments. *IFAC World Congress*, 44(1), 6842–6850.

Haddadin, S., Urbanek, H., Parusel, S., Burschka, D., Romann, J., Albu-Schaeffer, A., and Hirzinger, G. (2010). Real-time reactive motion generation based on variable attractor dynamics and shaped velocities. In *2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 3109 – 3116.

Ide, S., Takubo, T., Ohara, K., Mae, Y., and Arai, T. (2011). Real-time trajectory planning for mobile manipulator using model predictive control with constraints. In *2011 Int. Conf. on Ubiquitous Robots and Ambient Intelligence*, 244–249.

Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., and How, J. (2009). Real-time motion planning with applications to autonomous urban driving. *IEEE Trans. on Control Systems Technology*, 17(5), 1105–1118.

Ogren, P., Egerstedt, N., and Hu, X. (2000). Reactive mobile manipulation using dynamic trajectory tracking. In *2000 IEEE Int. Conf. on Robotics and Automation*, 3473–3478.

Oriolo, G. and Vendittelli, M. (2009). A control-based approach to task-constrained motion planning. In *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 297–302.

Qian, X., Navarro, I., de La Fortelle, A., and Moutarde, F. (2016). Motion planning for urban autonomous driving using Bezier curves and MPC. In *2016 IEEE Int. Conf. on Intelligent Transportation Systems*, 826–833.

Quirynen, R., Vukov, M., Zanon, M., and Diehl, M. (2015). Autogenerating microsecond solvers for nonlinear MPC: A tutorial using ACADO integrators. *Optimal Control Applications and Methods*, 36(5), 685–704.

Salaj, M., Gulan, M., and Rohal'-Ilkiv, B. (2015). Pendubot control scheme based on nonlinear MPC and MHE exploiting parallelization. In *2015 Int. Conf. on Intelligent Engineering Systems*, 353–358.

Vannoy, J. and Xiao, J. (2008). Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environment with unforeseen changes. *IEEE Trans. on Robotics*, 24(5), 1199–1212.