2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-7281-6212-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/IROS45743.2020.9341382

Autonomous task planning and situation awareness in robotic surgery*

Michele Ginesi, Daniele Meli, Andrea Roberti, Nicola Sansonetto and Paolo Fiorini

***Abstract*— The use of robots in minimally invasive surgery has improved the quality of standard surgical procedures. So far, only the automation of simple surgical actions has been investigated by researchers, while the execution of structured tasks requiring reasoning on the environment and the choice among multiple actions is still managed by human surgeons. In this paper, we propose a framework to implement surgical task automation. The framework consists of a task-level reasoning module based on answer set programming, a low-level motion planning module based on dynamic movement primitives, and a situation awareness module. The logic-based reasoning module generates explainable plans and is able to recover from failure conditions, which are identified and explained by the situation awareness module interfacing to a human supervisor, for enhanced safety. Dynamic Movement Primitives allow to replicate the dexterity of surgeons and to adapt to obstacles and changes in the environment. The framework is validated on different versions of the standard surgical training peg-and-ring task.**

I. INTRODUCTION

In the last decades, the use of robots in the operating room has provided help to surgeons in performing minimally invasive surgery, improving the precision of gestures and the recovery time for patients [1]. At present, surgeons teleoperate slave manipulators acting on the patient using a master console. One frontier of research in surgical robotics [2] is the development of a cognitive robotic system able to understand the scene and autonomously execute a (part of an) operation, emulating human reasoning and requiring gradually less monitoring from experts [3]. Increasing the level of autonomy could further improve the quality of an intervention, in terms of safety and patient recovery time [4]. Moreover, it could optimize the use of the operating room, solving issues such as surgeon fatigue and reducing hospital costs. The Artificial Intelligence (AI) community has widely investigated challenges towards autonomous (surgical) robotics, including real-time situation awareness for monitoring and adaptation of the surgical workflow, explainable plan generation for safety, dexterous trajectory generation and adaptation even in a small workspace. So far, most of the research in surgery has focused on the interpretation of sensors to guide simple actions, e.g. knot-tying [5] and drilling [6].

*This research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, ARS (Autonomous Robotic Surgery) project, grant agreement No. 742671.

Authors are with Department of Computer Science, University of Verona, Strada le Grazie, 15, 37134, Verona, Italy. {michele.ginesi, daniele.meli, andrea.roberti, nicola.sansonetto, paolo.fiorini}@univr.it

In this paper, we address the problem of the automation of a training surgical task, where multiple actions must be coordinated and sensor-driven cognition is needed to solve non-standard occurrences in the task. This covers level 2 of autonomy as of [3]. We use the da Vinci® robot from Intuitive Surgical as our testbed and we focus on a more complex version of the peg-and-ring task, where both single-arm and dual-arm executions are possible, depending on dynamic conditions on rings and pegs. We propose a framework which integrates Answer Set Programming (ASP) for task planning, and Dynamic Movement Primitives (DMPs) for trajectory generation and obstacle avoidance in real-time.
ASP is an explainable AI tool to reason on sensory information and on prior expert knowledge of the task. DMPs allow to emulate the dexterity of surgeons learning from a small dataset of gestures. Although the examined task is still far from real surgery, to the best of our knowledge this is the first framework that combines real-time situation awareness, adaptive task/motion planning with failure explanation, and recovery within a surgical setup.

The paper is organized as follows: in Section II we review the state of the art in surgical task automation and explainable task automation in robotics with safety concerns. In Section III we present the framework and the task, and in Section IV we show the experimental results. A short Section of discussion and future perspectives follows.

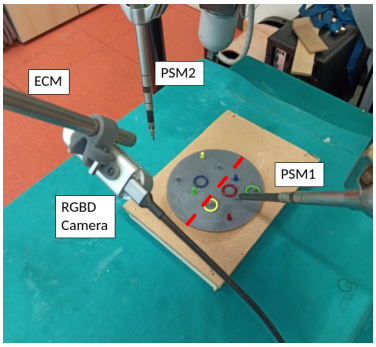


Fig. 1: The setup for the peg-and-ring task. The red dashed line defines reachability regions for the two arms.

II. STATE OF THE ART

Recent research has investigated the automation of basic operations in surgery, e.g. automated suturing with a precise mechanical needle guide [7] peg-and-ring and blunt dissection [8]. The automation of the peg-and-ring task has been proposed in [9], using RGBD camera for better accuracy. All mentioned works rely on a prior sequential description of the tasks, assuming a static environment and ignoring

reaction to anomalous events. A cognitive framework is proposed in [10], where a hidden Markov model (HMM) is used to monitor the execution of needle insertion and tune stiffness parameters for the admittance control to safely drive the instrument to the goal. Data-driven models as HMMs and neural networks are also used in robotic surgery for the interpretation of data from multiple sensors [11]. However, a huge amount of data, which is not usually available in surgery, is typically required by these techniques to achieve robust learning. Moreover, data-driven models are black-box, hence they generate plans that cannot be easily monitored by human experts [12], affecting the acceptability of the autonomous system in safety-critical surgery. Instead, we focus on knowledge-based reasoning systems, which encode prior expertise from humans and offer a clearer interpretation of the execution workflow. Knowledge representation for autonomous agents has been proposed in several robotic contexts [13], [14]. The most popular example is Knowrob [14], where an ontology represents general-purpose knowledge and a planning language [15] is used to query knowledge and define task specifications. In [16] we have proposed an ontology for the automation of the peg-and-ring task with an industrial manipulator. However, our experiments have evidenced the limits of ontologies, which can only reason on a static representation of the scenario, while real-time knowledge update for reactive planning is computationally inefficient. For this reason, ontologies have been mostly used as support to human monitoring in safety-critical applications, e.g. rehabilitation [17], and industry [18]. On the contrary, non-monotonic programming offers a more flexible framework for logic planning [19], allowing to update incomplete and dynamic knowledge with new observations. Examples in autonomous driving [20], aerospace [21] and industry [22] show the feasibility of non-monotonic reasoning in challenging safety-critical scenarios. While the most popular tool for non-monotonic planning is Prolog [23], we use the more recent framework of Answer Set Programming (ASP) [24] for its better computational efficiency and higher expressivity, allowing e.g. preference reasoning for optimal planning [25].

III. MATERIALS AND METHODS

A. The peg-and-ring task

The peg-and-ring task consists of placing rings on the same-colored pegs, using the two slave manipulators of the DVRK named PSM1 and PSM2 (see Figure 1 for our setup). We add extra specifications to the standard task description to increase the complexity and highlight the capabilities of the reasoner. Rings may either be grasped and placed by the same arm, or they can be transferred between arms, depending on the relative position of rings and pegs with respect to the center of the base. Pegs can be occupied by other rings, so they must be freed before placing another ring. Moreover, rings may not be present at the beginning, or they can either be on pegs or on the base, thus extraction may be required. The positions of rings can change in real-time, so the plan must be continuously adapted to the current environment, and the motion trajectories must reach moving goals. The dynamic nature of the scenario emulates a real patient's anatomy, and it requires the autonomous system to quickly adapt to new conditions and failure to reach the final goal. Even if peg-and-ring is not a proper surgical task, it is widely used as a training exercise for surgeons, since it presents several challenges in common with real surgery, and therefore we see it as a necessary first step before addressing more realistic tasks. In fact, the slave manipulators must move in a surgical-scale environment, avoiding obstacles (e.g., parts of the anatomy), grasping and positioning small objects with precision and dexterity (like needle grasping in suturing).

B. The framework

A scheme of our framework is shown in Figure 2. The exchange of information between its modules and the real system (robot + sensors) is shown. The flow of information towards an external human observer is also represented. The human can read the semantic conditions identified by sensors and the plan scheduled for execution at runtime, monitoring the correctness of the overall system. A description of the functions of each module and details about their integration follows.

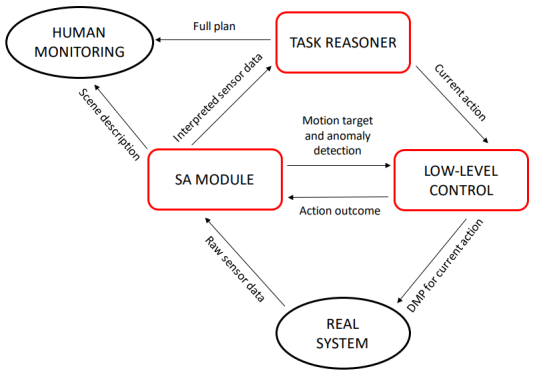


Fig. 2: The proposed framework for surgical task automation. Functional modules of the framework are highlighted in red, while arrows show the stream of information between modules, the real system and an external human observer.

1) Task reasoning module.: A task reasoner based on ASP is implemented in this module. An answer set program defines the entities and specifications of the task, in terms of Boolean variables (*atoms*) and logical implications on atoms (*rules*). Entities are the objects involved and the actions, while specifications describe the effects and pre-conditions of actions, task constraints, and the goal. For the peg-and-ring task, entities are the agents `Arm` (PSM1 and PSM2), the `ring` and the `peg` with their `Color` (red, green, blue, yellow and grey). Actions with pre-conditions and effects are defined as follows:

- `move(Arm, ring, Color)` to move to a colored ring, with pre-condition `reachable(Arm, ring, Color)` and effect `at(Arm, ring, Color)`;

- `move(Arm, peg, Color)` to move to a colored peg, with pre-condition `reachable(Arm, peg, Color)` and effect `at(Arm, peg, Color)`;
- `move(Arm, center)` to move to the transfer point, with pre-condition `in_hand(Arm, ring, Color)` and effect `at(Arm, center)`;
- `grasp(Arm, ring, Color)` to grasp a colored ring, with pre-condition `at(Arm, ring, Color)` and effect `in_hand(Arm, ring, Color)`;
- `release(Arm)` to open the gripper, with pre-condition `closed_gripper(Arm)` and effect `not in_hand(Arm, ring, Color)`;
- `extract(Robot, ring, Color)` to remove a colored ring from a peg, with pre-condition `in_hand(Arm, ring, Color)` and effect `not on(ring, Color1, peg, Color2)`.

The atom `reachable` states that a ring or a peg can be reached by an arm, depending on its relative position with respect to the center of the base. Some atoms are defined as *external*, namely they can be set by other programs, in order to allow integration with sensors. External atoms are `reachable`, `on`, `closed_gripper` and `in_hand`. Additionally, the atom `distance(Arm, ring, Color, Value)` is introduced to define the distance between rings and arms. This will be used for optimal plan generation in Section IV, exploiting pre-defined constructs for preference reasoning and optimization in ASP. External atoms are recognized by the situation awareness module at runtime. We also define executability constraints to implement user-defined specifications: a ring cannot be grasped by an arm with closed gripper; a ring that is on a peg cannot be moved before extraction; a ring cannot be placed on an occupied peg. These constraints emulate standard safety requirements in real surgery. The goal is defined as the constraint that all reachable rings are placed on their pegs, assuming `on(ring, Color, peg, Color)` is an effect of `at(Arm, peg, Color)`, `in_hand(Arm, ring, Color)`, `release(Arm)`.

Given this task description, the ASP Solving Algorithm (1) based on SAT solving [26] is executed. First, *grounding* of the external atoms as received from sensors is performed. This assigns an initial truth value to the corresponding Boolean variables. Then, the solver checks the grounded pre-conditions, and matches them with the effects of possible actions, incrementing a discrete-time step until the goal is satisfied. We assume a one-step delay between pre-conditions, actions, and effects. Finally, the sequence of actions which minimizes the time horizon to reach the goal is returned. Notice that the specifications do not determine a fixed temporal sequence of the actions as in standard FSMs, but they only define high-level task-related knowledge which must be taken into account by the ASP solver to produce the fastest feasible plan. By default, we use the *aggregate* construct (`0 { Action: Pre-condition } 1`) from ASP to force the solver to return at most one action per time step. However, in Section IV we will relax this constraint to `0 { Action: Pre-condition } 1 :- arm(Arm)`, which allows one action *per robot* at each time step. Therefore, the reasoner will automatically decide whether the arms should co-operate or act independently, reducing the time to complete the task.

Algorithm 1 ASP Solving Algorithm

```
Input: ASP program with specifications, external atoms
Output: Plan
Ground external atoms
Plan = [], t = 1, Action = null
while not goal do
    if Action != null then
        Ground effects of Action
    end if
    Check pre-conditions for actions at t
    if some actions are possible then
        Select Action with effect closest to goal
        Plan.append(Action(t))
        t++
    else return Unsatisfiable
    end if
end while
return Plan
```

2) Low-level control module: This module receives the actions computed by the task reasoner, and executes the corresponding low-level control policies in temporal sequence. For each `move` action, a trajectory for the specified arm and target is generated using DMPs [27]. DMPs consist of a system of second-order ODEs (one equation for each dimension of the ambient space) with a perturbation term. DMPs aim to model the perturbation term in such a way to be able to generalize the trajectory to new start and goal positions while maintaining the shape of the learned trajectory. The d-dimensional formulation is given by

$$\begin{cases} \tau\dot{\mathbf{v}} = \mathbf{K}(\mathbf{g}-\mathbf{x}) - \mathbf{D}\mathbf{v} - \mathbf{K}(\mathbf{g}-\mathbf{x}_0)s + \mathbf{K}\mathbf{f}(s) & \text{(1a)} \\ \tau\dot{\mathbf{x}} = \mathbf{v}. & \text{(1b)} \end{cases}$$

where $\mathbf{x}, \mathbf{v} \in \mathbb{R}^d$ are, respectively, position and velocity of a point (end-effector) of the system. Matrices $\mathbf{K}, \mathbf{D} \in \mathbb{R}_+^{d\times d}$ are diagonal ($\mathbf{K} = \text{diag}(K_1, K_2, \ldots, K_d)$, $\mathbf{D} = \text{diag}(D_1, D_2, \ldots, D_d)$) and satisfy the *critical damping condition* $D_i = 2\sqrt{K_i}$. $\mathbf{g}, \mathbf{x}_0 \in \mathbb{R}^d$ are, respectively, the goal and starting position, and $\tau \in \mathbb{R}_+$ is a time-scaling parameter. Function $\mathbf{f} : \mathbb{R} \to \mathbb{R}^d$ is the perturbation term. $s \in \mathbb{R}_+$ is a re-parametrization of time governed by the so-called *canonical system*,

$$\tau\dot{s} = -\alpha s, \quad \alpha \in \mathbb{R}_+,$$

with initial condition $s(0) = 1$.

During the learning phase, a desired trajectory $\widetilde{\mathbf{x}}(t) \in \mathbb{R}^d$, $t \in [0, T]$ is recorded. By fixing the elastic and damping parameters $K_i, D_i, i = 1, 2, \ldots, d$, and imposing $\tau = 1$ and $\mathbf{x}_0 = \widetilde{\mathbf{x}}(0)$, $\mathbf{g} = \widetilde{\mathbf{x}}(T)$, we can solve (1a) to compute the *desired forcing term* $\widetilde{\mathbf{f}}(s(t))$. Next, we approximate $\widetilde{\mathbf{f}}(s)$ in

each dimension using basis functions $\psi_i(s)$:

$$\widetilde{f}(s) \approx f(s) = \frac{\sum_{i=0}^{N} \omega_i \, \psi_i(s)}{\sum_{i=0}^{N} \psi_i(s)} \, s.$$

We use, as set of basis functions ψ_i, the *mollifier-like basis functions* proposed in [28]. The weights ω_i are computed so as to minimize the L_2-error between the desired and the approximated forcing term $\|\widetilde{f} - f\|_2$. Once the weights ω_i have been computed, (1) can be solved changing $\mathbf{x}_0$, and $\mathbf{g}$. The set of weights defines the low-level policy for the specific action to be executed. Moreover, by changing τ it is possible to change the speed of execution of the trajectory. To make the execution more robust against changes of starting and goal position, we use the approach presented in [28] to make DMPs invariant under rotation and dilatation of the relative position $\mathbf{g} - \mathbf{x}_0$.

In order to apply the DMPs framework to our task, we need to model also the orientation of the end-effector to replicate the dexterity of surgeons. To do so, we rely on the DMPs formulation in unit quaternion space presented in [29]. We emphasize that the same canonical system is 'shared' among Cartesian and orientation DMPs so that all the trajectories are synchronized.

Obstacle avoidance is implemented in the DMP framework by using the method proposed in [30], in which an obstacle is modeled as a repulsive potential field, whose negative gradient is added to (1a) to perturb the trajectory. In our scenario, obstacles are represented by pegs.

The execution of a DMP can be interrupted if an anomaly is detected by the situation awareness module, in which case a new plan generation is requested.

Algorithm 2 Vision Algorithm

Input: Point Cloud P_{in} in real time
Output: Poses of rings $Pose_{ring}$ and pegs $Pose_{peg}$
for t = 1 to ∞ **do**
  Subsample $P_{in}(t)$ *to* $P_{sub}(t)$
  if t = 1 **then**
    Plane estimation
    return $Pose_{peg}$
  else
    for colorID = 0 to 3 **do**
      Color Segmentation of $P_{sub}(t)$
      Euclidean clustering
      Ring identification $\leftarrow$ *RANSAC*
      return $Pose_{ring}(t)[colorID]$
    end for
  end if
end for

3) Situation Awareness (SA) module: This module is in charge of the semantic interpretation of data from sensors, providing a high-level real-time description of the environment in real-time which can be easily read from human supervisors and enhances explainability and safety of the framework. Moreover, the SA module acts as an intermediate layer between task- and motion-level modules, improving the scalability and generality of the framework. The inputs to the SA module are the real-time poses of pegs and centers of rings from a RGBD camera, and the poses of the arms from kinematics. These poses are computed with respect to a common frame *world* for the camera and the robotic arms using hand-eye calibration. During the execution of the task, the Vision Algorithm (2) subsamples the point cloud from the scene in order to guarantee real-time performances. The base and the pegs are assumed to be static during the whole execution, and they are identified only at the beginning of the task. The poses of all rings are retrieved at each time step. The identification of pegs and rings is performed in two steps. First, color segmentation allows to identify same-colored points. Then, Euclidean clustering allows to separate the clouds of ring and peg. Finally, Random Sample Consensus (RANSAC) is used to fit a torus shape on both clusters, and the best fitting cluster is identified as the ring, while the other as the peg.

Algorithm 3 SA Algorithm

Input: Action, $Pose_{ring}$, $Pose_{peg}$
Output: Failure message, target pose, external atoms
failure = True
if failure **then**
  Compute_externals($Pose_{ring}$, $Pose_{peg}$*)*
  return external atoms
  failure = False
else if Executing Action **then**
  while Action not ended **do**
    Compute_target($Pose_{peg}$*)*
    return target pose
    if Action = `move_ring` **then**
      if $Pose_{ring}[colorID]$ is not retrieved **then**
        failure = True
        return failure
      end if
    else if Action = `move_peg` **then**
      if ring fallen or peg occupied **then**
        failure = True
        return failure
      end if
    else if Action = `move_center` **then**
      if ring fallen **then**
        failure = True
        return failure
      end if
    end if
  end while
end if

The output of the vision algorithm is used by the SA Algorithm (3) to provide external atoms to the task reasoner, check failure conditions and compute targets for the low-level control module. Atoms and failure conditions are identified from geometric information retrieved from camera.

For instance, `in_hand(Arm, ring, Color)` is recognized when the distance between the ring and the arm is below a threshold and the gripper is closed, while the fall of the ring is recognized when the distance between the arm and the ring increases over a threshold during motion. When an action is started by the low-level controller, the specific failure conditions and target pose are computed during the whole execution. In detail, when moving to a ring, the grasping point is selected as the point of the ring cloud most distant from the pegs. Given the position r of a point on the ring and the set X of positions of pegs, the function to be maximized is $\sqrt{\sum_{p \in X} \|r-p\|_2^2}$. In this way, we guarantee collision-free grasping. Given the grasping point, the orientation is chosen such that the gripper approaches the ring orthogonally to the ring's plane. When moving to a peg, the target orientation is chosen orthogonal to the plane, so that the DMP can automatically recover in case the ring flips. Finally, during transfer between the PSMs, the target point for the free arm is chosen as the one opposite to the grasping point of the main arm. In case an anomaly is identified, the low-level control module is notified and the updated external atoms are sent to the task reasoner to compute a new plan.

IV. EXPERIMENTAL RESULTS

For the experimental evaluation of the framework, we use the da Vinci Research Kit (DVRK)[1]. The communication between modules of the framework relies on ROS infrastructure. The task reasoning module is implemented using the state-of-the-art grounder and solver Clingo [31], which offers Python APIs for easy integration with ROS, as well as useful tools for incremental time-horizon solving, definition of external atoms and optimization statements. We use an Intel RealSense d435 camera, which can see depth images from at least 0.105 m. The Point Cloud Library (PCL) is used as the standard tool to process the stream from the camera, offering integration with ROS and useful tools for RANSAC segmentation. We prefer a RGBD camera since the standard surgical endoscope has smaller baseline between the stereo cameras, reducing the depth range of view and degrading the localization accuracy. Moreover, our hand-eye calibration [32] with a marker on a custom calibration board (Figure 3a) allows to reach a state-of-the-art precision of 1.6 mm in pose detection, comparable with the intrinsic accuracy of the da Vinci® [33]. The RGBD camera is rigidly attached to the end-effector of the endoscopic arm of the da Vinci® (ECM) with a properly designed adapter.

The DMPs for `move` actions are learned from multiple human executions using the approach presented in [28]. Three users with different dexterity performed five trials each of the task in teleoperation. The initial positions of rings and pegs and the order of the rings (red, green, blue, yellow) are the same for all executions. Rings are always transferred between the arms. In this way, we get 120 executions of the `move(Arm, ring, Color)` gesture (at the beginning and during transfer for each ring), 60 executions of the `move(Arm, peg, Color)` and `move(Arm, center)` gestures. The learning process averages over all human trajectories, without any different weight or bias for executions from more expert users. Figure 4 shows the learned Cartesian DMP for the `move(Arm, ring, Color)` gesture as an example.

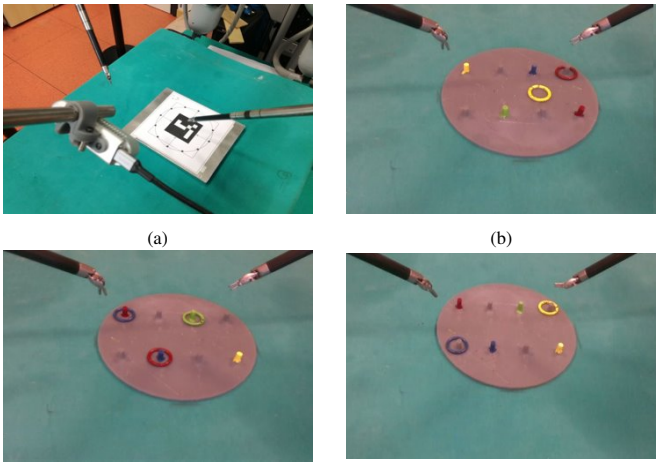
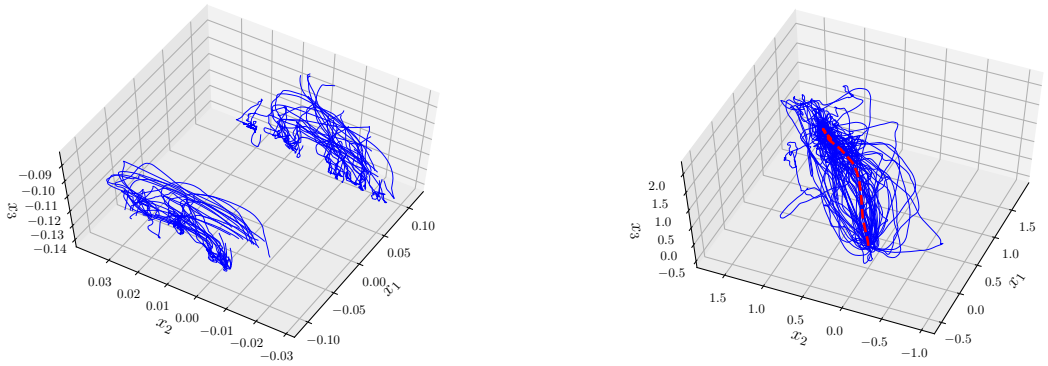


Fig. 3: Custom calibration board (top-left) and the tested scenarios as seen from the Realsense.

We validate our framework in three different scenarios, shown in the linked video and in Figure 3. In scenario 3b we show the main different versions of the peg-and-ring task: the red ring is placed on a grey peg and must be extracted, and the yellow ring requires transfer. In spite of the calibration accuracy, the small size of the setup and light conditions originate vision errors. In this scenario, the reasoner is also able to re-plan when the first grasping of the yellow ring fails. Figure 5 shows the main steps of the execution, highlighting the semantic scene interpretation and plan generation. In the video, we show a similar scenario with the blue and red rings, where the system is able to recover from a different failure condition when transfer fails. In scenario 3b, we exploit preference reasoning in ASP to perform optimization and take the closest ring (red) first, using `distance` variable defined in Section III. In scenario 3c, colored pegs are occupied, hence a ring must be temporarily placed on a grey peg. This operation is not encoded in the ASP program, but the ASP solver autonomously finds this solution as time-optimal from given constraints. Moreover, the SA module identifies the green ring as already placed, hence the reasoner ignores it. Finally, in scenario 3d we test the simultaneous execution of the two arms to complete the task faster, using ASP aggregates as described in Section III. In the video, we also show the grounded state variables from the SA module.

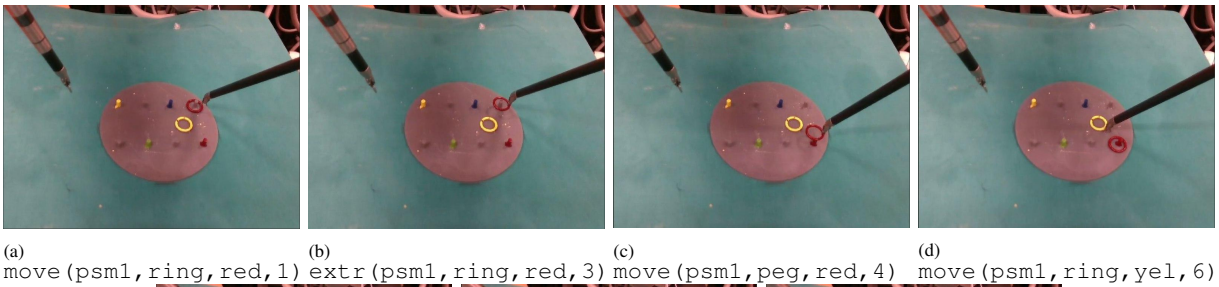
In Table I we show the task planning times for the tested scenarios. We also show the planning time for the standard scenario with all rings in the scene requiring transfer. This is the worst-case scenario, since more actions are needed to reach the goal. The results prove the real-time capabilities of

[1] `https://github.com/jhu-dvrk/sawIntuitiveResearchKit`



(a) (b)

Fig. 4: a) The set of Cartesian trajectories for the `move_ring` gesture, for both PSMs; b) trajectories after roto-dilatation, to start at the origin, $\mathbf{x}_0 = \mathbf{0}$ and end at the vector of ones, $\mathbf{g} = \mathbf{1}$ for batch learning. The learned DMP is shown in red dashed line.



(a) `move(psm1,ring,red,1)` (b) `extr(psm1,ring,red,3)` (c) `move(psm1,peg,red,4)` (d) `move(psm1,ring,yel,6)`

(e) `move(psm1,ring,yel,9)` (f) `move(psm1,center,11)` (g) `move(psm2,peg,yel,14)`

Fig. 5: Main actions of example execution in scenario 3b (`grasp`, `release` actions are omitted for simplicity). The initial plan is generated in a) after the grounding of the information from the SA module: `reachable(psm1,ring,red)`, `reachable(psm1,ring,yellow)`, `reachable(psm1,peg,red)`, `reachable(psm1,peg,blue)`, `reachable(psm2,peg,yellow)`, `reachable(psm2,peg,green)`, `on(ring,red,peg,grey)`. e) shows the re-planning of action `move(psm1,ring,yellow)` when *ring_fallen* is received from SA module.

our task planner (1.78 s in worst-case scenario). We notice that optimization increases the planning time.

TABLE I: Planning time for the ASP planner in the tested scenarios and in the worst-case scenario (complete) with all four rings requiring transfer.

	Planning time [s]
Scenario 3b (optimization)	0.108 (0.424)
Scenario 3c	0.133
Scenario 3d	0.113
Complete (optimization)	1.780 (8.636)

V. CONCLUSIONS

In this paper, a framework for the autonomous execution of surgical tasks was presented. This novel framework is the first fundamental step to address the problems of failure recovery, explainable plan generation, and situation awareness in the surgical scenario, which are required features for the acceptability of an autonomous surgical system. We have focused on a more complex version of the peg-and-ring task, a standard in the training program for surgeons, with the da Vinci® surgical robot. An ASP-based task reasoner is able to quickly coordinate the minimal set of actions in non-standard task conditions, integrating with semantic sensing of the scene and respecting task-specific constraints. Motion trajectories are learned from teleoperated executions by users with different expertise, using the DMP framework to replicate human dexterity.

One drawback of our framework is the use of a RGBD camera instead of a surgical endoscope to reach sufficient precision, hence the application to real surgery is still limited.

We will extend our calibration procedure to the standard endoscope in the future. Also, evaluation of different metrics to establish a benchmark with respect to human execution (e.g., smoothness of trajectories, reaction time to failure) and implementation of visual servoing to improve the repeatability of the system is part of our on-going research. We have been also investigating how to enrich ASP knowledge with semantic learning and external knowledge bases (e.g., surgical ontologies), and we will validate our framework on more surgically relevant tasks.

REFERENCES

[1] T. J. Vidovszky, W. Smith, J. Ghosh, and M. R. Ali, "Robotic cholecystectomy: learning curve, advantages, and limitations," *Journal of Surgical Research*, vol. 136, no. 2, pp. 172–178, 2006.

[2] G. P. Moustris, S. C. Hiridis, K. Deliparaschos, and K. Konstantinidis, "Evolution of autonomous and semi-autonomous robotic surgical systems: a review of the literature," *The international journal of medical robotics and computer assisted surgery*, vol. 7, no. 4, pp. 375–392, 2011.

[3] T. Haidegger, "Autonomy for surgical robots: Concepts and paradigms," *IEEE Transactions on Medical Robotics and Bionics*, vol. 1, no. 2, pp. 65–76, 2019.

[4] G.-Z. Yang, J. Cambias, K. Cleary, E. Daimler, J. Drake, P. E. Dupont, N. Hata, P. Kazanzides, S. Martel, R. V. Patel, *et al.*, "Medical robotics—regulatory, ethical, and legal considerations for increasing levels of autonomy," *Science Robotics*, vol. 2, no. 4, p. 8638, 2017.

[5] D.-L. Chow, R. C. Jackson, M. C. Çavuşoğlu, and W. Newman, "A novel vision guided knot-tying method for autonomous robotic surgery," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2014, pp. 504–508.

[6] C. Coulson, R. Taylor, A. Reid, M. Griffiths, D. Proops, and P. Brett, "An autonomous surgical robot for drilling a cochleostomy: preliminary porcine trial," *Clinical Otolaryngology*, vol. 33, no. 4, pp. 343–347, 2008.

[7] S. Sen, A. Garg, D. V. Gealy, S. McKinley, Y. Jen, and K. Goldberg, "Automating multi-throw multilateral surgical suturing with a mechanical needle guide and sequential convex optimization," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4178–4185.

[8] T. D. Nagy and T. Haidegger, "A dvrk-based framework for surgical subtask automation," *Acta Polytechnica Hungarica*, pp. 61–78, 2019.

[9] M. Hwang, D. Seita, B. Thananjeyan, J. Ichnowski, S. Paradis, D. Fer, T. Low, and K. Goldberg, "Applying Depth-Sensing to Automated Surgical Manipulation with a da Vinci Robot," in *International Symposium on Medical Robotics (ISMR)*, 2020.

[10] R. Muradore *et al.*, "Development of a cognitive robotic system for simple surgical tasks," *Int J of Advanced Robotic Systems*, vol. 12, no. 4, p. 37, 2015.

[11] Y. Kassahun, B. Yu, A. T. Tibebu, D. Stoyanov, S. Giannarou, J. H. Metzen, and E. Vander Poorten, "Surgical robotics beyond enhanced dexterity instrumentation: a survey of machine learning techniques and their role in intelligent and autonomous surgical actions," *International journal of computer assisted radiology and surgery*, vol. 11, no. 4, pp. 553–568, 2016.

[12] R. Goebel, A. Chander, K. Holzinger, F. Lecue, Z. Akata, S. Stumpf, P. Kieseberg, and A. Holzinger, "Explainable ai: the new 42?" in *International cross-domain conference for machine learning and knowledge extraction*. Springer, 2018, pp. 295–303.

[13] A. Loutfi, S. Coradeschi, M. Daoutis, and J. Melchert, "Using knowledge representation for perceptual anchoring in a robotic system," *International Journal on Artificial Intelligence Tools*, vol. 17, no. 05, pp. 925–944, 2008.

[15] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[14] M. Tenorth and M. Beetz, "Knowrob – a knowledge processing infrastructure for cognition-enabled robots," *International Journal of Robotics Research (IJRR)*, vol. 32, no. 5, pp. 566–590, 2013.

[16] M. Ginesi, D. Meli, H. C. Nakawala, A. Roberti, and P. Fiorini, "A knowledge-based framework for task automation in surgery," in *2019 19th International Conference on Advanced Robotics (ICAR)*, Dec 2019, pp. 37–42.

[17] Z. Dogmus, G. Gezici, V. Patoglu, and E. Erdem, "Developing and maintaining an ontology for rehabilitation robotics." in *KEOD*, 2012, pp. 389–395.

[18] S. Puls, J. Graf, H. Wörn, and I. Maurtua, "Cognitive robotics in industrial environments," in *Human Machine Interaction-Getting Closer*. InTech, 2012, pp. 213–234.

[19] Y. Dimopoulos, B. Nebel, and J. Koehler, "Encoding planning problems in nonmonotonic logic programs," in *European Conference on Planning*. Springer, 1997, pp. 169–181.

[20] M. Gebser, P. Obermeier, T. Schaub, M. Ratsch-Heitmann, and M. Runge, "Routing driverless transport vehicles in car assembly with answer set programming," *Theory and Practice of Logic Programming*, vol. 18, no. 3-4, pp. 520–534, 2018.

[21] M. Balduccini, M. Gelfond, R. Watson, and M. Nogueira, "The usa-advisor: A case study in answer set planning," in *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 2001, pp. 439–442.

[22] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, and E. C. Teppan, "Industrial applications of answer set programming," *KI-Künstliche Intelligenz*, vol. 32, no. 2-3, pp. 165–176, 2018.

[23] A. Colmerauer, "An introduction to prolog iii," in *Computational Logic*. Springer, 1990, pp. 37–79.

[24] V. Lifschitz, "Answer set planning," in *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 1999, pp. 373–374.

[25] G. Brewka, I. Niemela, and M. Truszczynski, "Preferences and nonmonotonic reasoning," *AI magazine*, vol. 29, no. 4, pp. 69–69, 2008.

[26] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll (t)," *Journal of the ACM (JACM)*, vol. 53, no. 6, pp. 937–977, 2006.

[27] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal, "Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance," in *Robotics and Automation, IEEE International Conference on*. IEEE, 2009, pp. 2587–2592.

[28] M. Ginesi, N. Sansonetto, and P. Fiorini, "Dmp++: Overcoming some drawbacks of dynamic movement primitives," *arXiv preprint arXiv:1908.10608*, 2019.

[29] M. Saveriano, F. Franzel, and D. Lee, "Merging position and orientation motion primitives," in *International Conference on Robotics and Automation (ICRA), 2019*, 2019.

[30] M. Ginesi, D. Meli, A. Calanca, D. Dall'Alba, N. Sansonetto, and P. Fiorini, "Dynamic movement primitives: Volumetric obstacle avoidance," in *2019 19th International Conference on Advanced Robotics (ICAR)*, Dec 2019, pp. 234–239.

[31] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, "A user's guide to gringo, clasp, clingo, and iclingo," 2008.

[32] A. Roberti, N. Piccinelli, D. Meli, R. Muradore, and P. Fiorini, "Improving rigid 3d calibration for robotic surgery," *arXiv preprint arXiv:2007.08427*, 2020.

[33] T. Haidegger, P. Kazanzides, I. Rudas, B. Benyó, and Z. Benyó, "The importance of accuracy measurement standards for computer-integrated interventional systems," in *EURON GEM Sig Workshop on The Role of Experiments in Robotics Research at IEEE ICRA*, 2010, pp. 1–6.