Six Degrees of Planning: Automated Planning for Surgical Navigation Under MyCobot's Six Degrees of Freedom

Colton Barr [*1], **Mateus Karvat Camara** [*1], **Sidney Givigi** [1]

{c.barr, mateus.karvat, sidney.givigi}@queensu.ca
[1] Queen's University, School of Computing, Kingston, Canada

Abstract

Over the past decade, medical robotics has continued to gain significant traction in a variety of surgical contexts. It has also enabled previously impractical procedures, such as stereo-electro-encephalography, to be performed at a speed and precision that improves upon current standards of care. The commercial robotics systems that exist to support this procedure, however, are typically difficult to interface with for research purposes. This limits the ability of AI researchers to explore promising new robotics paradigms, such as the application of automated planning within robotic motion planning. With the introduction of low-cost research-grade robotics platforms, such as MyCobot's 6 Degrees of Freedom Collaborative Robotic Arm, as well as new software bridges between Robot Operating System and popular medical research platforms, there is an emerging interest in using lower-cost robotic development platforms to research procedures like stereo-electro-encephalography placement. In this paper, we demonstrate the use of automated planning techniques to perform motion planning on a low-cost 6 Degrees of Freedom robotic arm for the purposes of stereo-electro-encephalography placement. We use Hybrid Planning via the Expressive Numeric Heuristic Search Planner to model the individual joints of a MyCobot arm that has a needle as an end effector. Each joint has functions that describe the [x,y,z] coordinates, angle and length of the joint, with two core actions for moving the joint and one for stopping its motion. The problem setting includes information about the location of the target entry point as a goal state, as well as the position of the patient's head as a forbidden region. The performance of the planner was then tested in different scenarios and configurations to understand how it performs under a variety of problem settings. Further optimizations were performed to the problem and domain specification to improve the quality and safety of the plans that were generated.

Introduction

Surgical navigation broadly refers to the practice of utilizing information beyond that which is immediately accessible to the surgeon's senses to provide spatial information during a procedure. This can range in sophistication from viewing static medical images during a procedure, akin to a sailor reading a paper map, to the use of an external 3D measurement system for real-time tool localization, like a modern ship equipped with GPS. Recent advances in medical robotics and AI have made surgical navigation platforms that utilize robotics increasingly popular, favoured for their improved ergonomics, speed, and precision over some traditional approaches (Goh and Ali 2022). One such example of a procedure that has benefited from the recent development of novel surgical robotics solutions is stereo-electro-encephalography (SEEG).

SEEG is a neurosurgical procedure that involves the placement of electrodes at precise 3D coordinates within the brain in order to record epileptic seizure activity and localize potential targets for surgery (Chassoux et al. 2018). While this method has considerable advantages over other less invasive monitoring techniques which suffer from substantially reduced spatial accuracy, it requires a high degree of precision to place the electrodes deep within the brain safely. Until recently, the only spatial measurement technique with sufficient accuracy for the procedure was the use of mechanical guides manually configured for each electrode during the procedure. Over the course of between 12-15 electrodes, this method resulted in prohibitively long operative times. With the introduction of robotic navigation for SEEG, the transition between electrode insertion sites can be significantly reduced while achieving similar or improved electrode placement accuracy (Mullin, Smithason, and Gonzalez-Martinez 2016).

Over the past two decades there have been several robotics platforms introduced to support neurosurgical procedures like SEEG (Faraji, Remick, and Abel 2020), including the ROSA stereotactic robot (Lefranc et al. 2014), the Neuromate Robot (Kajita et al. 2015), and the Mazor Renaissance system (Liang et al. 2022). These systems include a variety of safety mechanisms and fail-safes to ensure that they do not deviate from their planned trajectories, and come equipped with planning software to specify no-go zones and areas where the arm must reduce speed. The clinicians are required to specify their target point and trajectory, as well as forbidden regions, and the robotic platforms utilize their own planning software to specify the kinematics of the arm. While the specifics of these motion planning algorithms are proprietary, they would need to account for the unique state space that includes the current tool, the electrode trajectories and the regions specified by the user. They might also

[*]These authors contributed equally.

include additional objective functions to minimize the time for a given motion or improve motion accuracy by favouring smaller motions, just as the Neuromate system reported to account for the optimal working range of the joint encoders for a given platform (Kajita et al. 2015).

Robotic motion planning is a complex problem, with a well-established field built over the past several decades dedicated to solving it. With the rapid pace of AI advancement over the past few years there has been increasing interest in the applications of AI to robotic motion planning, typically through the field of automated planning. A major development was the introduction of ROSPlan in 2016 (Cashmore et al. 2015), enabling the direct application of Planning Domain Definition Language (PDDL) to robotic motion planning. In this project, we aim to explore the applications of automated planning to medical robotics by implementing a domain and problem in PDDL to simulate SEEG placement with MyCobot's 6 Degrees of Freedom (DOF) Collaborative Robotic Arm.

By considering a needle attached to the robotic arm's end, our goal is to find a sequence of joint rotations that take the tip of the needle to a specified target location without hitting the patient's head during the trajectory, either with the needle or the arm itself. Using Hybrid Planning, we employ the Expressive Numeric Heuristic Search Planner (ENHSP) (Scala et al. 2016) with movement actions happening through time and states representing the 3D positioning of each of the joints of the robot. The following sections will detail related works in this space, the implementation of our planning model, evaluation of the model within a custom 3D Slicer simulation (Kikinis, Pieper, and Vosburgh 2014), and a discussion of the limitations and next steps for this project.

Related Work

Robotic motion planning refers broadly to the problem of determining the series of movements or trajectories that a robot must take to achieve a goal or perform a specific task (Masehian and Sedighizadeh 2007). The planning tools used to solve robotic motion problems can range from traditional graph and path planning algorithms, such as A* or Dijkstra's algorithm, to supervised learning techniques like Monte Carlo search tree and recurrent neural networks, to more recent advancements in reinforcement learning (Zhou, Huang, and Fränti 2022). Given the wide range of robotic motion planning applications, robotic platforms and agent embodiments that exist, frequently motion planning strategies are not developed to be widely generalizable.

In the field of robotic motion planning for multi-link robotic manipulator arms, the focus of this paper, this trend is especially true. The specific geometry and applications of a given robotic platform, at least at a commercial level, tend to be built into the motion planning solution the platform employs, limiting the scalability and generalizability of a given motion planning solution (Bertolucci et al. 2019). The prospect of using a universal planning language to articulate a given problem and domain in robotic motion planning, and apply state-of-the-art planners to the problem at hand, has driven increasing interest in applications of automated planning within the field. An early project in this domain was ROSPlan, a framework for using planning tools and Planning Domain Definition Language (PDDL) in combination with the Robot Operating System (ROS) library (Cashmore et al. 2015). While this project opened up the tools of planning to the wider ROS community, some limitations of the base language (PDDL) may have reduced its wider applicability. Garrett et al. extended the functionality of PDDL for robotic motion planning in their paper PDDLStream, where streaming objects add sampling procedures to help robots with sensing capabilities (Garrett, Lozano-Pérez, and Kaelbling 2018), while Castro et al. built the Task and Motion Planning framework around an extended version of PDDL (Castro 2022). Both of these frameworks are well suited to straddling levels of abstraction in higher-level task planning and lower-level robotic motion planning but still suffer from the numerical calculation limitations of PDDL. The continuous processes and events introduced in PDDL+ tend to simplify the modelling of temporal planning domains and have gained traction in recent years for robotic manipulation and path planning. Along these lines, Thomas et al. found that PDDL+ was better suited than using PDDL for their hybrid planning framework for task and motion planning (Thomas et al. 2019), and Belolucci et al. demonstrated the application of exogenous events triggered by the environment to help model a dual-arm robot (Bertolucci et al. 2019).

While there have been significant advances in open-source solutions to robotic motion planning, and numerous commercial surgical robotics solutions that have been introduced over the past decade, there still does not exist, to the best of our knowledge, an accessible platform for open-source development of robotic navigation solutions using automated planning techniques. Moreover, we are unaware of any papers to date that explore the application of planners for path planning in surgical robotics. The fields of image-guided therapy and medical robotics have historically suffered from siloed approaches, exacerbated by differences in standard programming tools and challenges related to cross-platform compatibility of ROS and IGT tools. The recent introduction of Slicer-ROS2, however, combined with advancements in accessible robotics platforms, such as MyCobot, has made the concept of accessible open-source medical robotics development more feasible than ever (Connolly et al. 2022). In this paper, we demonstrate the application of automated planning in the context of robotic SEEG placement, and in doing so hope to highlight the emerging capacity for accessible robotic image-guided therapy research.

Model Description

A hybrid planning approach was selected for this problem given the need to keep track of numeric variables that change over time. These numeric variables must be updated over time to reflect the position of each joint of the robot as it moves, and the movements themselves must be updated to avoid collisions.

Then, to track the position of each of the joints of MyCobot's 6 DOF Collaborative Robotic Arm (which we will be referring to as "the robot" from now on) and simulate the effects of its movements, we created the model presented in Figure 1. Joints are named J_i, with i ranging from 0 to 9, and movements are named M_j, with j being equal to the number of the first joint that will be affected by its movement (e.g. M_2 is so called because neither J_0 nor J_1 would change their positions due to its movement, but all the joints with $i \geq 2$ would). In this model, segment $\overline{J8J9}$ corresponds to the needle. Even though a direction is shown for each movement in Figure 1, all of the movements can happen in any rotational direction: clockwise or counterclockwise.

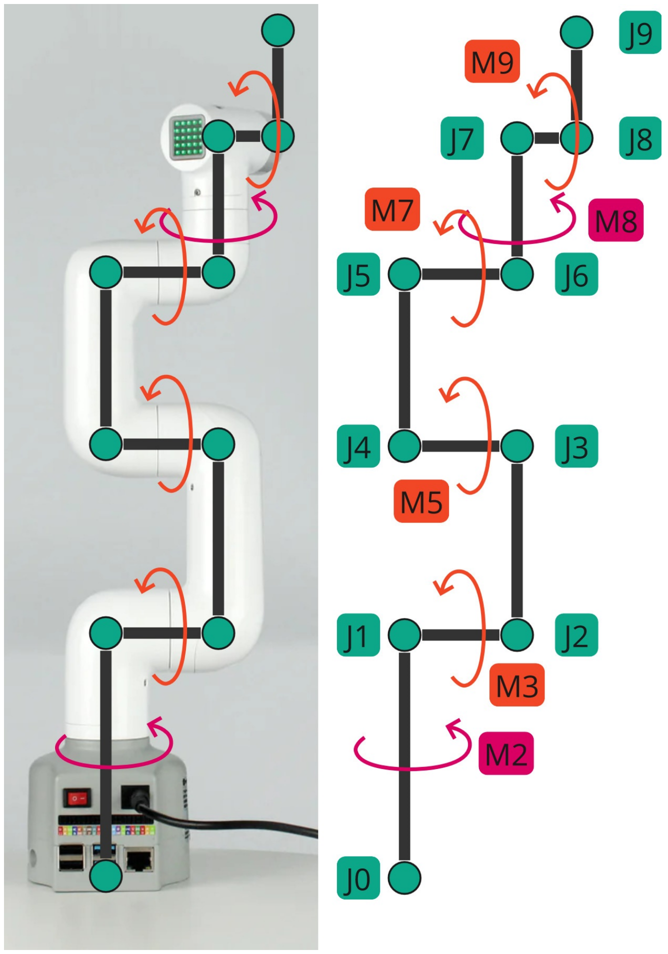

Figure 1: MyCobot's 6 DOF Collaborative Robotic Arm and the model created to simulate the movement of its joints. Movements are shown in a single direction to avoid cluttering the representation, but they all happen bi-directionally. Source of MyCobot's image: (ElephantRobotics 2023).

Since each M_j movement is angular, it's paramount to keep track of the angles between joints. For that matter, we consider that each movable joint (J_2, J_3, J_5, J_7, J_8 and J_9) starts in the configuration presented in Figure 1, with all angles set to 0. Clockwise movements reduce the value of the angle, while counterclockwise movements increase it. As such, Figure 2 presents an example of a movement in which $\theta = \pi$ (180°) and also the aforementioned consideration that, for a movement M_j, only joints J_i with $i \geq j$ will move.

In order to decrease the computational cost required to calculate the positions of the joints during movements, we follow a mixed angle representation, in which some of the joints have absolute angles and some of them have relative angles (Capitanelli et al. 2018). J_2, J_3, J_5 and J_7 have absolute angles (i.e. if J_3 increases its angle by θ, J_7 will also do so) while J_8 and J_9 have relative angles (i.e. only their own movement will change the value of their angles).

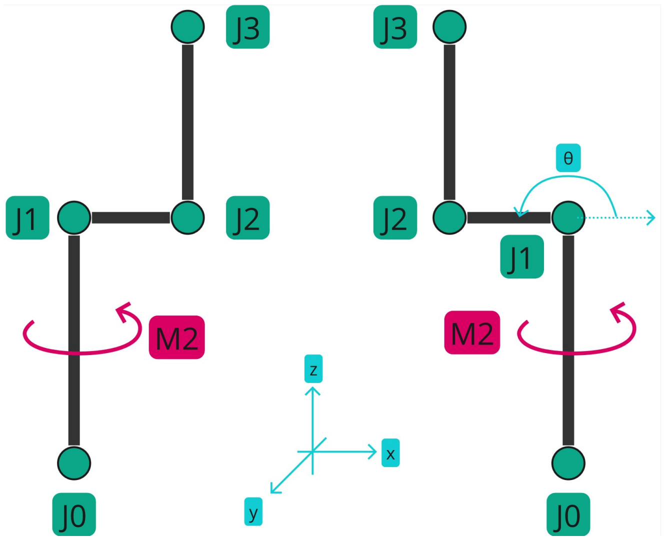

Figure 2: Partial representation of the robot, modelling an angular movement of +180 degrees.

Our real-world problem has the robot move the needle towards a target while avoiding the head. Therefore, besides the robot, our model also has an obstacle which, for the sake of simplifying collision calculations, is described as a sphere.

The model implementation is available on GitHub [1].

Initial state

The initial state is comprised of a set of fluents that describe the positions and configurations of all the objects in the problem. Thus, for every movable joint J_i, there are 5 functions set to initial values, shown in Code 1 (non-movable joints have the same functions, except for angle). While `ji_x`, `ji_y` and `ji_z` describe the initial x, y and z positions of joint J_i, `ji_angle` describes its angle, and `li` refers to the length of this segment.

```
(= (ji_x) 5.0)
(= (ji_y) 0.0)
(= (ji_z) 3.0)
(= (ji_angle) 0.0)
(= (li) 5.0)
```

Code 1: Example of joint functions for a joint J_i.

Besides the joint functions, there are global functions that are used to describe the problem setting, shown in Code 2.

[1]https://github.com/MahatKC/6DOP

target_x, target_y and target_z are related to the target's 3D position.

```
(= (target_x) -5.0)
(= (target_y) 5.0)
(= (target_z) 3.0)

(= (sphere_center_x) 4.0)
(= (sphere_center_y) 4.0)
(= (sphere_center_z) 3.0)
(= (squared_sphere_radius) 9.0)

(no_movement)
(= (lambda) 100)
(= (w) 0.0174533)
(= (updating_positions) 0)
(= (squared_joint_radius) 4)
```

Code 2: Global functions related to the problem setting with arbitrary values.

Then, sphere_center_x, sphere_center_y and sphere_center_z refer to the obstacle's center, with squared_sphere_radius being the square of the sphere's radius. Since the sphere will always have the same dimension throughout the execution of the plan, its radius is described as the square radius to avoid having to calculate it for every single collision verification.

The no_movement fluent is set to True since no movement is initially happening, lambda is the acceptable squared error (which will be better described in Subsection) and w is the angular velocity of the joints' movements (in radians), which is set to 0.0174533 to allow for better visualization of the plan, since 0.0174533 radians are equal to $1°$ (i.e. if a joint moves for 90 seconds, it has moved $90°$).

Finally, updating_positions is a flag used to force the planner to first update the angles and then the positions of the joints (Subsection describes its related event) and squared_joint_radius is used to model collisions between joints.

Varying experimental conditions may be modelled by changing the values of the target position (target_x, target_y and target_z), head position (sphere_center_x, sphere_center_y and sphere_center_z) and head size (squared_sphere_radius), considering a fixed initial position for the robot. However, some experimental conditions might lead to unsolvable problems (e.g. target outside of the reach of the robot), for which the value of lambda may be increased leading to a viable solution.

Actions

In a broad sense, only two actions are used in the domain: move and stop. However, as can be seen in Figure 1, there are 6 different move actions, which have accompanying stop actions, and each move is actually broken down in two: move_clockwise and move_counterclockwise.

Therefore, a move_clockwise action for a joint J_i happens as shown in Code 3. Here, some boolean fluents related to the movements are used so that a few restrictions are imposed:

1. no two joints are moving simultaneously,
2. a joint only moves in a single direction at a time,
3. a joint J_i will only move after the previous joint J_{i-1} has finished moving.

```
(:action move_ji_counterclockwise
  :parameters ()
  :precondition (and
    (no_movement)
    (not (joint_i_moving))
    (not (joint_i_finished))
    (joint_i-1_finished)
  )
  :effect (and
    (joint_i_moving)
    (joint_i_moving_counterclockwise)
    (not (no_movement))
))
```

Code 3: Move action for a joint J_i.

The third restriction is necessary due to the limitations of ENHSP and PDDL+. For a joint J_i to move after a joint J_k, with $k > i$, it would be necessary to calculate the angle between J_k and J_{i-1}, which is only possible with the $arcsin()$ and $arccos()$ functions, which are currently not available in these tools.

The stop action then, shown in Code 4, will only change the boolean fluents that direct movement in the system. A single stop action is used for both movement directions. By working only with the boolean fluents, the actual numeric fluents (functions) can be manipulated exclusively by processes and events.

```
(:action stop_ji
  :parameters ()
  :precondition (and
    (joint_i_moving)
    (not (no_movement))
  )
  :effect (and
    (not (joint_i_moving))
    (not (joint_i_moving_clockwise))
    (not (
   joint_i_moving_counterclockwise))
    (no_movement)
    (joint_i_finished)
))
```

Code 4: Stop action for a joint J_i.

Processes

The actual angular movement of the robot's joints only happens with processes. Each move action has a corresponding moving process and the processes only start after their corresponding actions are performed. Any process for a joint J_i will cease its execution when a stop action for the joint J_i is performed. In Code 5, the movement booleans set up by the `move` action are preconditions for the process, which then updates the relevant angles according to the angular velocity defined in the problem.

```
(:process moving_ji_clockwise
  :parameters ()
  :precondition (and
    (joint_i_moving)
    (joint_i_moving_clockwise)
    (not (no_movement))
    (<= updating_positions 0)
  )
  :effect (and
    (decrease (ji_angle) (* #t w))
    (assign (updating_positions) 1.0)
))
```

Code 5: Moving process for a joint J_i.

After updating the angles, the `updating_positions` flag is raised, so the `update_positions` event will occur and the process will only resume after the event has had its effect. In an initial implementation, we updated the positions of the joints within processes, however that led to major issues regarding the order of operations performed by the planner. Since all of the calculations for the positions require the updated angle value, we left the angle update within the processes and moved the position updates to events, which solved those issues.

Events

As mentioned at the end of the previous section, the actual update of positions happens with the `update_positions` event, shown in Code 5. Given the characteristics of the robot, any M_j movement will change the 3D position of all joints J_i with $i \geq j$, so that the `update_positions` event for a joint will update the position of all joints after it (and including itself). Positions are only updated after the angle of joint J_i has been properly updated in the `moving` process. Then, after all positions are updated, the flag is set to zero so that the process may resume.

Collisions are handled by events, for which three collision events are considered: `head_collision`, `joint_collision` and `floor_collision`. If any collision happens, the effect is that all movement is stopped, taking the planner to a dead-end, since it is within the goal (described in Subsection) to not have any collisions whatsoever.

The `head_collision` event, as shown in Code 7, calculates, for a joint J_i, if the joint's 3D position is within the boundaries of the sphere. Here, the choice of setting the squared radius of the sphere instead of the radius becomes clear: calculations would get costly if done otherwise.

The `joint_collision` event checks, for a joint J_i, if it is colliding with all the other joints J_k that it might collide with while moving. The verification for collision is done in a similar fashion as the head collision, but instead of using the `squared_sphere_radius`, joint collision considers the `squared_joint_radius`, as can be seen in Code 8.

```
(:event update_positions_ji_moving
  :parameters ()
  :precondition (and
   (joint_i_moving)
   (= updating_positions 1.0)
  )
  :effect (and
   (assign (ji_x)
    (+ jk_x
      ;;calculated relative position in x
      axis
    )
   )
   (assign (ji_y)
    (+ jk_y
      ;;calculated relative position in y
      axis
    )
   )
   (assign (ji_z)
    (+ jk_z
      ;;calculated relative position in z
      axis
    )
   )
   ;; Updates all joints until the last
   one
   ;; ...
   (assign updating_positions 0.0)
))
```

Code 6: Update positions event for a joint J_i. The positions of the previous joint, J_k ($k = i - 1$) are used as reference to calculate the positions of the succeeding joints.

```
(:event head_collision_ji
  :parameters ()
  :precondition (and
    (not (no_movement))
    (or
      (<=
        (+
          (^ (- ji_x sphere_center_x) 2)
          (+ (^ (- ji_y sphere_center_y)
    2)
          (^ (- ji_z sphere_center_z) 2)
          )
        )
        (squared_sphere_radius)
      )
    )
  )
  :effect (and
    (no_movement)
    ;For k in {2,3,5,7,8,9}
    (not (joint_k_moving))
    (head_hit)
))
```

Code 7: Head collision event for a joint J_i.

Having a much simpler math behind it, the `floor_collision` event is presented in Code 9

```
(:event joint_collision_ji
  :parameters ()
  :precondition (and
    (not (no_movement))
    ;Ji&Jk Collision
    (<=
      (+
        (^ (- ji_x jk_x) 2)
        (+ (^ (- ji_y jk_y) 2)
        (^ (- ji_z jk_z) 2)
        )
      )
      (squared_joint_radius)
    )
    ;Check for all other Jk joints
    ;that Ji might collide with
  )
  :effect (and
    (no_movement)
    ;For k in {2,3,5,7,8,9}
    (not (joint_k_moving))
    (joint_hit)
))
```

Code 8: Joint collision event for a joint J_i

and just forbids any of the moving joints to have a negative value of z.

```
(:event floor_collision
  :parameters ()
  :precondition (and
    (not (no_movement))
    (or
      (<= j2_z 0.0)
      (<= j3_z 0.0)
      (<= j4_z 0.0)
      (<= j5_z 0.0)
      (<= j6_z 0.0)
      (<= j7_z 0.0)
      (<= j8_z 0.0)
      (<= j9_z 0.0)
    )
  )
  :effect (and
    (no_movement)
    ;For k in {2,3,5,7,8,9}
    (not (joint_k_moving))
    (floor_hit)
))
```

Code 9: Floor collision event.

In addition to collision events, there are also the angle reset events. Presented in Code 10 for any joint J_i, the angle reset sets the value of the angle back to 0 if it becomes 2π or -2π. This was created to reduce the search space of the planner to the set of angles between $(-2\pi, 2\pi)$ and help it converge to a solution in a finite length of time.

Goal

Our goal then is to take the last joint (J_9) to the target, which can be seen in Code 11. Here, we consider the square

```
(:event reset_ji_angle
  :parameters ()
  :precondition (and
    (or
      (>= (ji_angle) 6.283)
      (<= (ji_angle) -6.283)
    )
  )
  :effect (and
    (assign (ji_angle) 0.0)
))
```

Code 10: Reset angle event for a joint J_i.

of the distance of the last joint to the target and define the goal as having that square distance below a threshold value `lambda`. Even having 6 degrees of freedom, the robot would not be able to get the needle to every single point in space within its reach, but it would be able to get the needle extremely close to a target point, and that amount is determined by `lambda`. If the target can be reached, it is still useful to have a `lambda` value greater than 0 to reduce the search space of the planner and allow it to arrive at a viable plan within a reasonable time.

```
(:goal
  (and
    (<=
      (+ (^ (- j9_x target_x) 2)
         (+ (^ (- j9_y target_y) 2)
            (^ (- j9_z target_z) 2)
         )
      )
      (lambda)
    )
    (no_movement)
    (not (head_hit))
    (not (floor_hit))
    (not (joint_hit))
))
```

Code 11: Goal of the plan.

Besides getting the last joint to the target, the goal specifies that no collisions are allowed (`head_hit`, `floor_hit` and `joint_hit`), which makes the planner avoid the obstacle at all costs. Lastly, it is required that no movement must be happening, which in turn requires a `stop` action to be performed so that the plan can be successful.

Results

We managed to successfully implement and model the domain problem in PDDL+ so that, for a given problem, the ENHSP planner is capable of generating a sequence of angle rotations that should be performed by the MyCobot robot.

In order to optimize the performance of the planner and measure the quality of the planned motion, we performed a series of experiments. These included testing a series of planner heuristics, variants, and search strategies to minimize the number of states explored, evaluating the quality of the planner in 5 simulated planning domains, and visualizing the plan embodied in a physical MyCobot robot.

Planner Optimization

Since we are only using a single planner, ENHSP (Scala et al. 2016), we explored a variety of different planner configurations and monitored the changes in the number of states explored by the planner. The tested configurations, with their descriptions taken from ENHSP's website (Scala et al. 2016), were:

- **Heuristics:**
  - aibr: Additive Interval-Based Relaxation;
  - hadd: Additive version of sub-goaling heuristic;
  - hradd: Additive version of sub-goaling heuristic plus redundant constraints;
  - hmax: Hmax for Numeric Planning;
  - hrmax: Hmax for Numeric Planning with redundant constraints;
  - hmrp: heuristic based on MRP extraction;
  - blcost: goal sensitive heuristic;
- **Planner configurations:**
  - sat-hmrp: Sat planning with Greedy Best First Search plus MRP heuristic;
  - sat-hmrph: Sat planning with Greedy Best First Search plus MRP heuristic with helpful actions;
  - sat-hmrphj: Sat planning with Greedy Best First Search plus MRP heuristic with helpful actions and helpful transitions;
  - sat-hadd: Sat planning with Greedy Best First Search with numeric hadd;
  - opt-blind: baseline blind heuristic (gives 1 to state where goal is not satisfied and 0 otherwise) with A* search;
- **Search strategy:**
  - gbfs: Greedy Best First Search ($f(n) = h(n)$);
  - WAStar: WA* ($f(n) = g(n) + h_w * h(n)$);
  - wa_star4: WA* ($f(n) = g(n) + 4 * h(n)$).

The problem setting for these experiments was a single target coordinate for the end-effector that required only 3 joint movements but was qualitatively found to require substantial search from the planner to reach a solution. Since the runtime can vary based on CPU load at the time of planner execution, the number of nodes and states explored for a given configuration was reported as a compute-agnostic search metric.

Table 1 details the experiments performed to compare different heuristic methods available in the ENHSP implementation. Note that planner execution time was generally cut off after 5 minutes of computation, and values that include a $>$ indicate that a given configuration did not finish executing. This experiment indicated that many of the tested heuristics performed identically, including the default heuristic.

Heuristics	**Nodes**	**States**
aibr	623,730	1,245,663
hadd	623,730	1,245,663
hradd	623,730	1,245,663
hmax	>8,736,907	>17,449,625
hrmax	>8,731,711	>17,439,248
hmrp	623,730	1,245,663
blcost	>10,378,528	>10,378,620

Table 1: All heuristics that are available for use with ENHSP, and the corresponding number of nodes and states explored in trying to find a solution.

In Table 2, the results from testing distinct planner configurations on the same base problem setting are shown. These planners' configurations combine various heuristics with either SAT planning, which uses greedy best-first search, or optimal planning using A*, as indicated by the "sat" or "opt" prefix. Note that in general all planning configurations performed similarly, with the exception of two that were unable to find a solution within 5 minutes of execution.

Planner	**Nodes**	**States**
sat-hmrp	623,730	1,245,663
sat-hmrph	623,730	1,245,663
sat-hmrphj	623,730	1,245,663
sat-hadd	>5,742,012	>11,464,556
opt-blind	>6,909,179	>9,439,545

Table 2: The nodes and states explored using various ENHSP planner configurations.

Table 3 shows the results of testing out different search strategies available in ENHSP. The default search strategy, "gbfs", was the only tested strategy that was able to converge on a solution within the aforementioned time constraint.

Search strategy	**Nodes**	**States**
gbfs	623,730	1,245,663
WAStar	>6,660,941	>13,213,731
wa_star_4	>6,633,001	>13,208,743

Table 3: The number of explored nodes and states when ENHSP is used for planning the same problem setting using different search strategies.

Simulation Testing

In order to better model the planning context and visualize the output of the planner, a custom 3D Slicer extension was developed that enables users to define the patient position, forbidden region around the patient, and target point for the end effector. This extension also can be used to run the planner, visualize the resulting robotic path, and load the individual joint angles to physically test the plan using the MyCobot platform. Using this extension, we designed and ran a case study for a particular patient position and target location, with the objective of moving the end effector as close as possible to the target position. Across a series of lambda values, we found the closest the end effector was able to get to the target was 3.5 mm. Figure 3 shows a visualization of the generated plan in 3D Slicer, with the forbidden region around the patient displayed as a wireframe sphere and the target position visible in green.

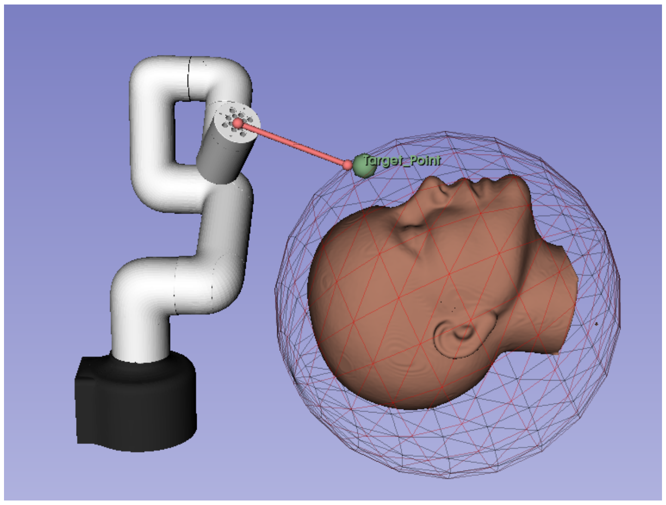

Figure 3: A visualization of the generated plan in the 3D Slicer extension, with the robot geometry received from Slicer-ROS2 visible in white, the target point in green, and the forbidden region around the patient's head as a wireframe sphere. Note that the end effector of the robot is modelled as a line and is shown in red.

Summary

In this paper, we demonstrated the application of automated planning for robotic motion planning in the context of neurosurgical robotics, using ENHSP for planning and 3D Slicer for problem simulation. Path planning of the MyCobot 6DOF robot platform was modelled as a hybrid planning problem using PDDL+, with the objective of moving the tip of a linear instrument to a prescribed target location without colliding with the patient's head. This planning model was integrated into a custom 3D Slicer application to simulate the problem, execute the planner and visualize the resulting robot path.

Future Work

There are several immediate next steps required to fully model the problem of robotic motion planning using the MyCobot 6DOF platform. These include modelling link collisions during planning, enabling simultaneous movement of separate joints, and allowing more complex paths that involve multiple movements of the same joint over time. Another simplification we added is that joints must move in order of closest to further from the base, which could be removed through additional modelling.

Another area of exploration would be to develop custom heuristics to be used with the ENHSP planner. These could be specifically tailored to the context of robotic motion planning and help simplify the search space for the planner, potentially improving runtime performance. In regards to the planner, an extension of ENHSP and PDDL+ would also be a possible avenue for future work, integrating the mathematical operators currently unavailable and modifying the goal definition to optimize safety constraints and resource usage.

Besides these improvements, further experiments could be performed comparing our approach to search-based motion planning methods, as well as an inverse kinematics solver combined with a continuous collision checker. Finally, equipping the custom 3D Slicer application to send joint positions to the robot following user previewing and approval would help make this system an end-to-end demonstration of executing an automated robotic surgical plan using 3D Slicer, a significant achievement for a low-cost configuration.

References

Bertolucci, R.; Capitanelli, A.; Maratea, M.; Mastrogiovanni, F.; and Vallati, M. 2019. Automated Planning Encodings for the Manipulation of Articulated Objects in 3D with Gravity. In Alviano, M.; Greco, G.; and Scarcello, F., eds., *AI*IA 2019 – Advances in Artificial Intelligence*, 135–150. Cham: Springer International Publishing. ISBN 978-3-030-35166-3.

Capitanelli, A.; Maratea, M.; Mastrogiovanni, F.; and Vallati, M. 2018. On the manipulation of articulated objects in human–robot cooperation scenarios. *Robotics and Autonomous Systems*, 109: 139–155.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. *Proceedings of the International Conference on Automated Planning and Scheduling*, 25(1): 333–341.

Castro, S. 2022. Integrated Task and Motion Planning (TAMP) in robotics - Robohub.

Chassoux, F.; Navarro, V.; Catenoix, H.; Valton, L.; and Vignal, J.-P. 2018. Planning and management of SEEG. *Neurophysiologie Clinique*, 48(1): 25–37. French Guidelines on Stereoelectroencephalography (SEEG).

Connolly, L.; Deguet, A.; Leonard, S.; Tokuda, J.; Ungi, T.; Krieger, A.; Kazanzides, P.; Mousavi, P.; Fichtinger, G.; and Taylor, R. H. 2022. Bridging 3D Slicer and ROS2 for Image-Guided Robotic Interventions. *Sensors*, 22(14).

ElephantRobotics. 2023. Raspberry Pi Robots: myCobot Pi 6-DoF collaborative robot from Elephant Robotics.

Faraji, A. H.; Remick, M.; and Abel, T. J. 2020. Contributions of Robotics to the Safety and Efficacy of Invasive Monitoring With Stereoelectroencephalography. *Frontiers in Neurology*, 11: 570010.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2018. STRIPStream: Integrating Symbolic Planners and Blackbox Samplers. *CoRR*, abs/1802.08705.

Goh, E. Z.; and Ali, T. 2022. Robotic surgery: an evolution in practice. *Journal of Surgical Protocols and Research Methodologies*, 2022(1): snac003.

Kajita, Y.; Nakatsubo, D.; Kataoka, H.; Nagai, T.; Nakura, T.; and Wakabayashi, T. 2015. Installation of a Neuromate Robot for Stereotactic Surgery: Efforts to Conform to Japanese Specifications and an Approach for Clinical Use—Technical Notes. *Neurologia medico-chirurgica*, 55(12): 907–914.

Kikinis, R.; Pieper, S. D.; and Vosburgh, K. G. 2014. *3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support*, 277–289. New York, NY: Springer New York. ISBN 978-1-4614-7657-3.

Lefranc, M.; Capel, C.; Pruvot, A. S.; Fichten, A.; Desenclos, C.; Toussaint, P.; Le Gars, D.; and Peltier, J. 2014. The Impact of the Reference Imaging Modality, Registration Method and Intraoperative Flat-Panel Computed Tomography on the Accuracy of the ROSA® Stereotactic Robot. *Stereotactic and Functional Neurosurgery*, 92(4): 242–250.

Liang, A. S.; Ginalis, E. E.; Jani, R.; Hargreaves, E. L.; and Danish, S. F. 2022. Frameless Robotic-Assisted Deep Brain Stimulation With the Mazor Renaissance System. *Operative neurosurgery (Hagerstown, Md.)*, 22: 158–164.

Masehian, E.; and Sedighizadeh, D. 2007. Classic and Heuristic Approaches in Robot Motion Planning - A Chronological Review. *World Academy of Science, Engineering and Technology*, 29.

Mullin, J. P.; Smithason, S.; and Gonzalez-Martinez, J. 2016. Stereo-Electro-Encephalo-Graphy (SEEG) With Robotic Assistance in the Presurgical Evaluation of Medical Refractory Epilepsy: A Technical Note. *Journal of Visualized Experiments : JoVE*, 2016: 53206.

Scala, E.; Haslum, P.; Thiebaux, S.; and Ramirez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*, ECAI'16, 655–663. NLD: IOS Press. ISBN 9781614996712.

Thomas, A.; Amatya, S.; Mastrogiovanni, F.; and Baglietto, M. 2019. Task-assisted motion planning in partially observable domains. *arXiv preprint arXiv:1908.10227*.

Zhou, C.; Huang, B.; and Fränti, P. 2022. A review of motion planning algorithms for intelligent robots. *Journal of Intelligent Manufacturing*, 33: 387–424.