PART C

# 1. Task description and motivation

This project focuses on programming a UR5e robotic arm to perform a structured manipulation task: picking up individual blocks and arranging them on a surface to visually spell out the letters "A" and "I". The experiment is conducted in the Mujoco simulation environment using a custom setup in the CLAIR lab framework.

This task was chosen because it captures the essence of symbolic goal execution in robotics using motion and spatial reasoning to achieve a visually meaningful result. Spelling out letters is more than just a low-level manipulation routine; it requires the robot to follow a deliberate, coordinated plan that transforms a high-level objective (the word “AI”) into a series of precise movements.

This task provides a solid platform to demonstrate key aspects of robotic planning and control. And also serves as a basic foundation for exploring cooperative tasks, symbolic communication, and high-level task planning in robotics.

# 3. Environment and Setup

- Simulation Framework: Mujoco + CLAIR lab UR5e setup

- Robotic System: Single UR5e arm

- Objects: 20 identical blocks

- Start Positions: Blocks initialized on a fixed grid

- Target Configuration: Blocks are placed at coordinates that render the characters "A" and "I"

# 6. Planning Strategy

Our planning process combined both symbolic and geometric reasoning. The goal of spelling “AI” was translated into specific target coordinates for each block, chosen to represent the letters clearly while staying within the robot’s workspace. These placements also took into account the size of the blocks and the need for consistent spacing between them.

To move each block from its initial position to the layout, we generated a sequence of movements designed to minimize the risk of collisions. Each motion involved lifting the block vertically, translating to the new location, and lowering it gently into place. While the paths were predefined, they were tailored to each block’s relative position to ensure smooth execution.

Instead of working directly in joint space, we planned the motions in task space and used inverse kinematics behind the scenes to compute valid arm configurations. This made it easier to focus on the spatial arrangement of the blocks and maintain a consistent grasp and placement behavior.

The planning also had to adapt to physical limitations of the robot and the simulation environment. For example, slight spacing was added between blocks to avoid unintended contact during placement, and block positions were adjusted to remain within reachable bounds.

# 7. Implementation strategy

The project was implemented in a Mujoco simulation using the UR5e robotic arm setup from the CLAIR lab framework. The simulation environment was configured with a grid of 20 blocks placed at predefined source positions. A separate set of target coordinates was designed to represent the layout of the letters “A” and “I”.

The logic was implemented in Python, using a custom script that loops through each block, sending it through a motion sequence: pick, lift, move, and place. The robot's movements were handled through existing motion planning utilities provided in the CLAIR codebase, allowing us to specify poses and rely on inverse kinematics for solving the corresponding joint configurations.

The system used simple modular functions for each stage of the motion, making the code flexible and easy to debug. Additionally, video recordings of the execution were taken directly from the simulation.

# 8. Challenges and Constraints

While the overall task was straightforward in scope, there were several practical limitations we had to consider. The simulation environment supported a maximum of 22 blocks, which meant we needed to use larger blocks to represent the letters “A” and “I” without compromising their readability. This required thoughtful layout adjustments to make the best use of the available space.

During placement, we observed occasional slight misalignments between blocks. These were likely due to minor inaccuracies in the robot’s control or actuation. To avoid blocks interfering with each other, we introduced small gaps between them in the target configuration. This simple adjustment helped maintain the stability and clarity of the final structure.

The robot’s motion was also constrained by the size and layout of the table, as well as by the physical limitations of its joints and angles. In some cases, target positions that were too far from the base of the robot couldn’t be reliably reached. To ensure all motions were feasible, we had to carefully place both the starting grid and the final layout within the robot’s reachable workspace.

# 10. Task Planning and algorithmic description

The task involved arranging 20 blocks to spell the letters “A” and “I” within a reachable and organized area of the workspace. The layout was designed to be compact, spreading across two rows and incorporating both horizontal and vertical elements to clearly form the two characters. Each letter was manually mapped to specific target coordinates, and each block started from a known position in a grid.

To automate the assignment between start and target positions, we implemented a greedy algorithm that matches each block to its nearest available target. For each block, the robot computes the 2D Euclidean distance to all unassigned targets and selects the closest one. This approach balances simplicity with spatial efficiency and ensures a clean, readable structure with minimal block movement.

The final motion plan consists of a loop where, for each block:

1. A pick-up motion is executed from the start grid.
2. The closest unassigned target is selected.
3. The block is moved and placed at that target with a slight vertical offset to ensure safe placement.
4. The target is marked as used to avoid reuse.

This structure allowed for minimal overlap between paths, maintained visual clarity, and supported consistent execution using the MotionExecutor utility from the CLAIR framework.

# 10. Future Improvements

- Implement collision detection and path validation

- Add vision-based feedback (e.g., camera to detect success)

- Extend to cooperative manipulation (two arms)

- Use randomized block initialization for testing planning flexibility

-Add obstacles and obstacle avoidance