MAE 204 — Final Project Report
Mobile Manipulation with the youBot

Your Name Here

March 18, 2026

1 Summary

This project implements software to control a youBot mobile manipulator performing a pick-and-place task: driving to a cube, grasping it, and placing it at a target location. The software is organized into four components:

1. **NextState** — a first-order Euler simulator that takes the current 12-variable robot configuration and the commanded joint/wheel velocities, then returns the configuration one timestep later. Chassis odometry follows the exact body-twist integration from Chapter 13.4, and joint/wheel speeds are clipped to a configurable maximum.

2. **TrajectoryGenerator** — produces an eight-segment reference trajectory for the end-effector using `ScrewTrajectory` from the Modern Robotics library. The segments are: (1) move to standoff above the initial cube, (2) descend to grasp, (3) close gripper, (4) return to standoff, (5) move to standoff above the goal, (6) descend to place, (7) open gripper, (8) return to standoff.

3. **FeedbackControl** — implements the feedforward-plus-PI feedback control law (textbook Eq. 11.16 and 13.37):

$$\mathcal{V}(t) = [\mathrm{Ad}_{X^{-1}X_d}]\,\mathcal{V}_d + K_p\,X_{\mathrm{err}} + K_i \int_0^t X_{\mathrm{err}}\,dt$$

   where $X_{\mathrm{err}} = \log(X^{-1}X_d)$. The commanded twist $\mathcal{V}$ is then mapped to wheel and arm joint speeds through the pseudoinverse of the 6×9 mobile-manipulator Jacobian $J_e(\theta)$.

4. **Wrapper script** (`run.py`) — ties everything together. It generates the reference trajectory, then loops through each pair of reference configurations, calling `FeedbackControl` and `NextState` at each step. The output is a CSV file playable in CoppeliaSim Scene 6.

Two design choices are worth noting:

Singularity avoidance. The end-effector approaches the cube at a 45° angle from vertical rather than perfectly horizontal. Early testing revealed that a horizontal approach put the arm near full extension at the grasp pose, creating a near-singular Jacobian whose pseudoinverse produced unreasonably large joint speeds. Tilting the approach keeps the arm in a better-conditioned configuration. I also set the pseudoinverse tolerance to `rcond` $= 10^{-3}$ so that small singular values are treated as zero, preventing large entries in $J_e^\dagger$ while still letting the controller act along near-singular directions when beneficial.

Self-collision avoidance. Because our task-space controller has no awareness of the robot's physical body, certain cube placements caused the arm to fold backward over the chassis. Inspecting the joint angles in the offending CSV files revealed joints well beyond their physical limits (e.g., $\theta_4 = -3.8\,\text{rad}$ when the limit is $\pm 1.79\,\text{rad}$), confirming self-collision. The root cause was the place-pose geometry: a cube goal orientation of $+90°$ forced the arm base joint θ_1 past $100°$, directing the arm toward the rear of the chassis, while the shoulder and wrist joints exceeded their ranges to compensate. The practical solution was twofold: (i) choose task configurations whose approach directions keep the arm in its forward workspace, and (ii) use the $45°$ tilted grasp so the arm need not fully extend horizontally. CoppeliaSim Scene 6 does not enforce joint limits or check self-collision, so these precautions must be taken at the software level. I also investigated implementing the Jacobian-column-zeroing method recommended in the MR Capstone reference, where columns of J_e corresponding to joints near their limits are set to zero before computing $J_e^\dagger$. While effective at preventing limit violations, this approach reduced the arm's reachable workspace enough that the gripper could no longer descend to the cube, causing all grasps to fail. A less restrictive variant (clamping only at the physical limits) similarly degraded tracking at the grasp and place poses. Given these trade-offs, the current implementation relies on the pseudoinverse tolerance and careful task-geometry selection to avoid both singularities and self-collision.

Note on plots. In all error and manipulability plots below, the time axis represents the elapsed simulation time in seconds ($\Delta t = 0.01\,\text{s}$ per step). The eight trajectory segments occupy roughly: 0–4 s (move to standoff), 4–5 s (descend to grasp), 5–5.6 s (close gripper), 5.6–6.6 s (lift), 6.6–10.6 s (transport to goal), 10.6–11.6 s (descend to place), 11.6–12.2 s (open gripper), and 12.2–13.2 s (lift).

2 Results: Best

- **Controller:** Feedforward + proportional (P) control.
- **Gains:** $K_p = 3I, \quad K_i = 0.$
- **Initial configuration:** $(\phi, x, y) = (\pi/6,\ -0.3,\ 0.1)$, arm joints $(0,\ -0.2,\ 0.3,\ -1.2,\ 0)$. This gives $40.1°$ orientation error and 0.337 m position error from the start of the reference trajectory.
- **Cube:** initial $(1, 0, 0)$, goal $(0, -1, -\pi/2)$.
- **Video:** INSERT_BEST_VIDEO_LINK_HERE

All six error components converge smoothly and monotonically to zero within roughly two seconds (Figure 1). There is no overshoot. The error remains essentially zero through the grasp, transport, and place phases. This is the expected behavior of a pure proportional controller: the error decays exponentially with a time constant of approximately $1/K_p \approx 0.33\,\text{s}$, which is fast enough to converge well before the grasp at $t \approx 5\,\text{s}$.

The manipulability plot (Figure 2) shows $\mu_1(A_\omega)$ settling near 0.31 after the initial transient, while $\mu_1(A_v)$ drops to around 0.1 during the grasp approach when the arm extends toward the cube. It recovers during the transport phase when the arm retracts.

3 Results: Overshoot

- **Controller:** Feedforward + proportional-integral (PI) control.

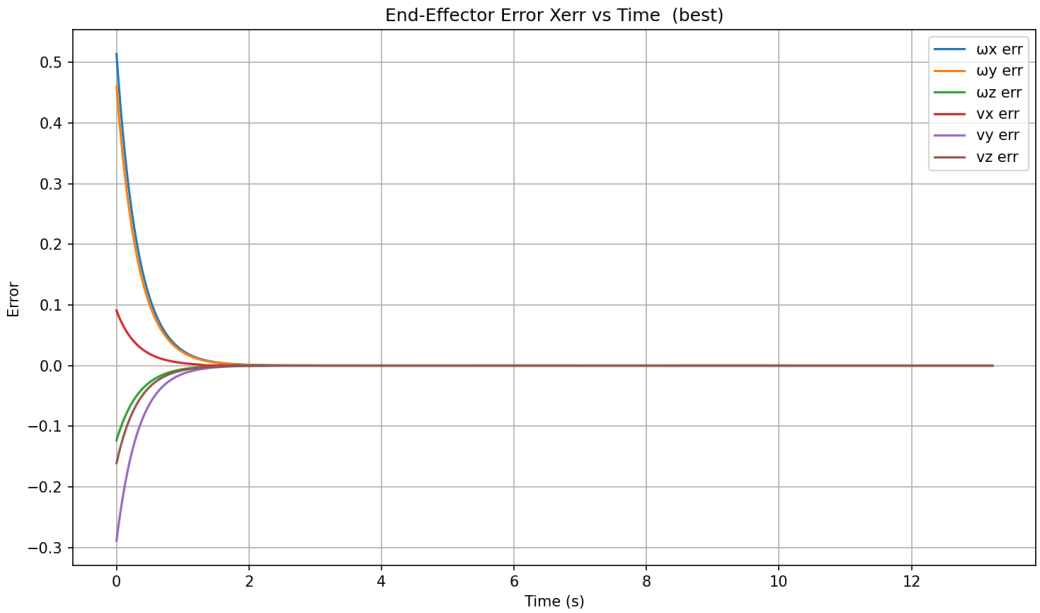

Figure 1: Best case — end-effector error X_{err} vs. time.

- **Gains:** $K_p = 2I, \quad K_i = 8I.$
- **Initial configuration:** Same as the best case.
- **Cube:** initial $(1, 0, 0)$, goal $(0, -1, -\pi/2)$.
- **Video:** INSERT_OVERSHOOT_VIDEO_LINK_HERE

Figure 3 shows clear overshoot and oscillation in the error, particularly in the ω_x, ω_y, and v_y components. After the initial rapid decay, each component crosses zero and oscillates for two to three cycles before settling. The error is effectively eliminated by roughly $t = 4\,\text{s}$ (end of trajectory segment 1), so the grasp still succeeds.

The oscillation is caused by the integral term: $K_i = 8I$ aggressively accumulates past error, so by the time the error reaches zero the integral has built up a large corrective signal that pushes the end-effector past the reference. This is the classic underdamped response of a PI controller.

The manipulability plot (Figure 4) is very similar to the best case because the same trajectory and initial configuration are used. Slight differences during 0–3 s reflect the different arm configurations visited due to the oscillatory transient.

4 Results: New Task

- **Controller:** Feedforward + PI control.
- **Gains:** $K_p = 5I, \quad K_i = 1I.$
- **Initial configuration:** $(\phi, x, y) = (0.3,\ -0.3,\ -0.3)$, arm joints $(0,\ -0.2,\ 0.3,\ -1.2,\ 0)$. This gives 31.9° orientation error and 0.240 m position error.
- **Cube:** initial $(0.5,\ -0.6,\ 0)$, goal $(1.0,\ 0.5,\ -\pi/4)$.

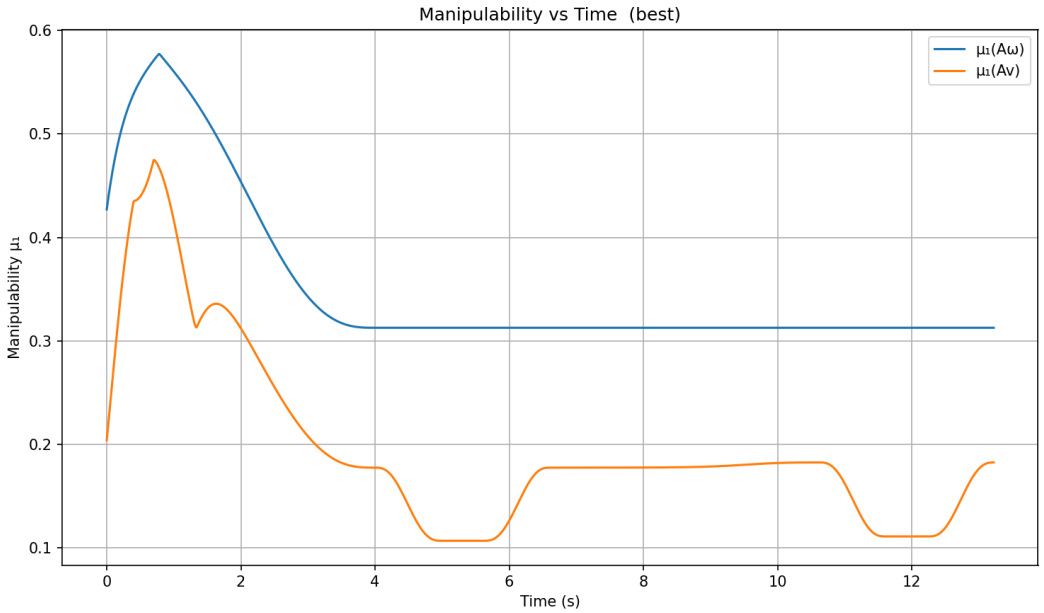

Figure 2: Best case — manipulability $\mu_1(A_\omega)$ and $\mu_1(A_v)$ vs. time.

- **Video:** INSERT_NEWTASK_VIDEO_LINK_HERE

This task picks up the cube from the right side of the workspace and places it at a $-45°$ rotation in the forward-left area—a qualitatively different geometry from the default scenario. The cube configurations are loaded from an external YAML file (`newtask_config.yaml`), making it easy to test other placements without modifying the code. I chose moderate gains ($K_p = 5I$, $K_i = 1I$) since the initial error is smaller than the default cases.

As shown in Figure 5, all six error components decay monotonically. The ω_y component starts largest ($\sim$0.47) and converges within the first second; the remaining components follow a slower exponential tail, reaching near-zero by $t \approx 4\,\text{s}$. The convergence is smooth with no overshoot, confirming that the PI gains are well-tuned for this configuration.

The manipulability plot (Figure 6) follows a pattern similar to the default scenarios: $\mu_1(A_\omega)$ peaks around 0.57 early on, then settles near 0.31, while $\mu_1(A_v)$ dips to $\sim$0.11 during the grasp and place approach phases when the arm is near full extension.

5 Discussion

1. **Advantages and disadvantages of the integral term; why it causes overshoot.** The integral term eliminates steady-state error that a proportional controller alone cannot remove—for example, bias from modeling errors or persistent disturbances. The downside is that the integrator accumulates error over time, so when the actual error reaches zero the accumulated integral is still nonzero and continues to drive the system past the reference. This causes the overshoot visible in the overshoot case. The system then oscillates as the integral "unwinds." Higher K_i values make this worse because the integral grows faster.

2. **When manipulability became small or large, and why.** $\mu_1(A_\omega)$ is largest ($\sim$0.57) early in the trajectory when the arm is in a compact, well-conditioned configuration near

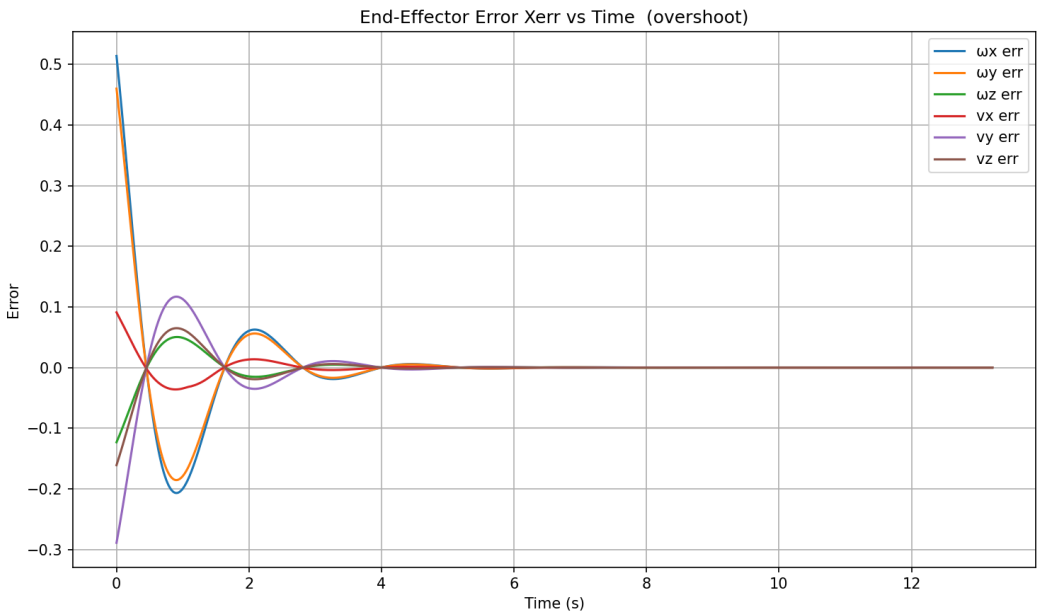

Figure 3: Overshoot case — end-effector error X_{err} vs. time.

the chassis. It decreases as the arm extends outward and settles around 0.31 for the rest of the motion. $\mu_1(A_v)$ is smallest ($\sim$0.1) during the grasp and place approach phases, when the arm is nearly fully extended to reach the cube. At near-full extension the arm is close to a kinematic singularity, so the smallest singular value of the linear velocity portion of the Jacobian approaches zero relative to the largest. During transport (segment 5), the arm retracts and manipulability recovers.

3. **Why error increases after picking up the cube when max joint velocities are low.** After grasping, segment 5 requires the robot to transport the cube from the initial standoff to the final standoff—a large reconfiguration of both the chassis and the arm. If joint and wheel velocities are severely limited, the robot physically cannot move fast enough to keep up with the reference trajectory. The feedforward twist $\mathcal{V}_d$ demands speeds that exceed the limits, so the clipped commands under-drive the robot, and tracking error accumulates.

4. **A task where velocity control is realistic, and one where it is not.** Velocity control is realistic for pick-and-place operations in a structured environment where the robot moves through free space and the main concern is following a geometric path accurately. It is *not* sufficient for tasks that require controlling interaction forces, such as polishing a surface or inserting a peg into a tight hole. In those cases the robot needs force/torque feedback to regulate contact forces, which velocity control alone cannot provide.

5. **Additional information needed for torque control.** To compute joint torques instead of joint velocities, `FeedbackControl` would need the full dynamic model of the robot: the mass matrix $M(\theta)$, the Coriolis/centrifugal terms $c(\theta, \dot{\theta})$, and the gravity vector $g(\theta)$. It would also need the current joint velocities $\dot{\theta}$. The corresponding MR function is `InverseDynamics` (or equivalently the computed-torque / inverse-dynamics controller from Chapter 11), which uses the Newton–Euler recursive algorithm to compute the required torques for a desired joint acceleration trajectory.

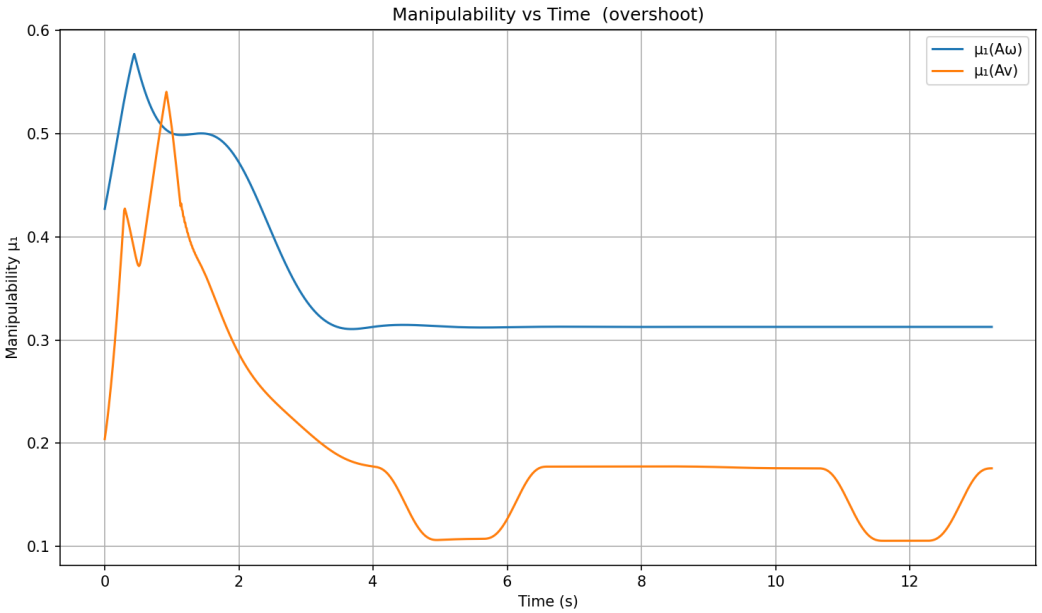

Figure 4: Overshoot case — manipulability vs. time.

6. **Use of generative AI.** Yes. I used Cursor (an AI-assisted code editor powered by a large language model) to help implement the four components, debug the grasp-singularity issue, generate plots, and draft portions of this report. All code was reviewed and tested by me, and I verified correctness against the test values provided in the project description.

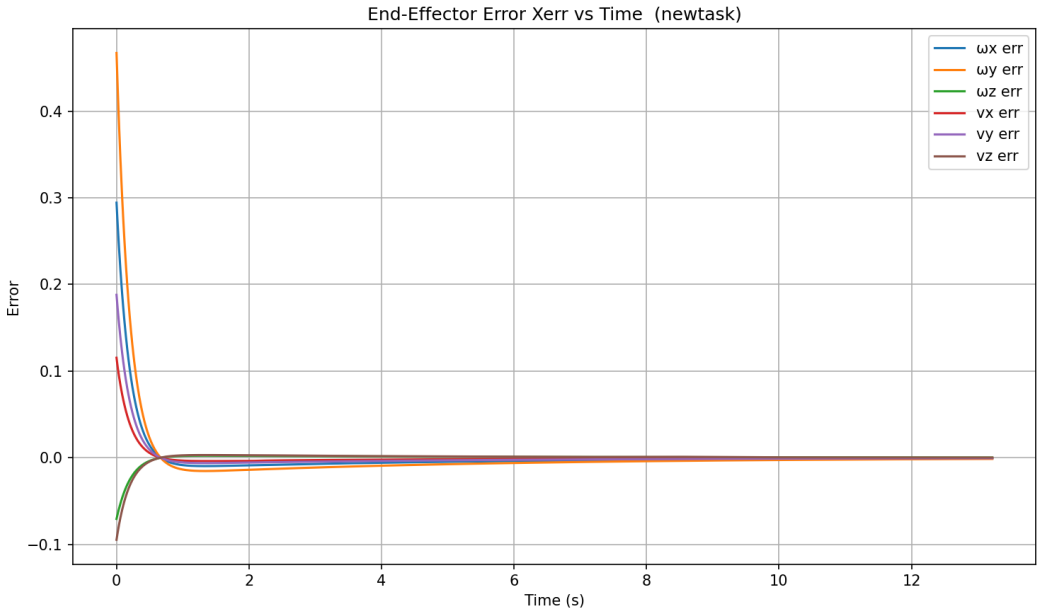

Figure 5: New task — end-effector error X_{err} vs. time.

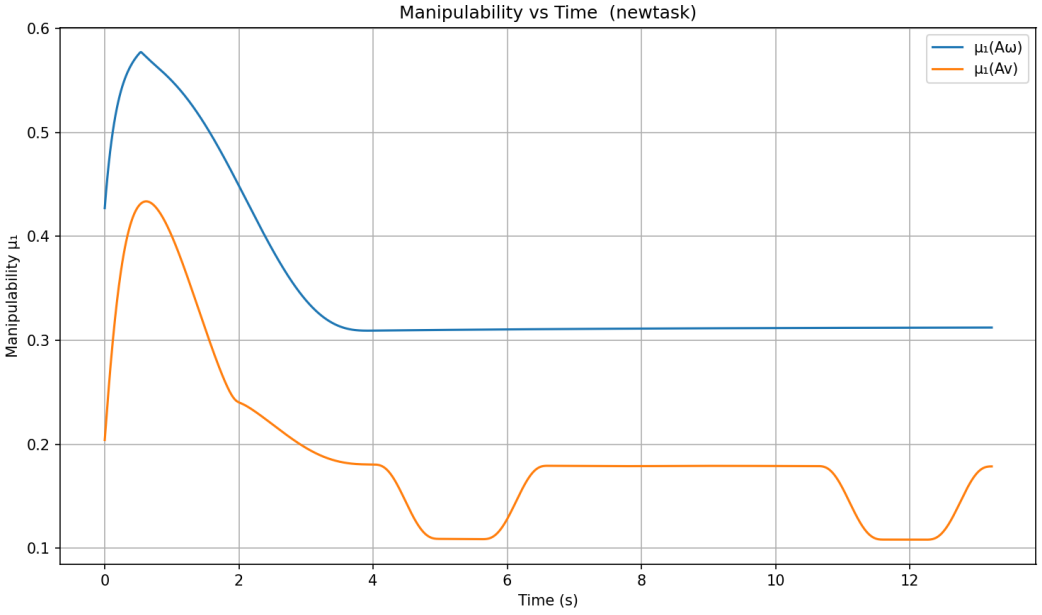

Figure 6: New task — manipulability vs. time.