A3: Mobile Manipulation

Benjamin Kuipers and Tiancheng Ge

Assigned: October 11, 2019
Challenge: October 28, 2019, 9 am, in class
Report Due: October 28, 2019, 11:59 pm

Overview

The goal of this assignment is for you to master robot arm manipulation, object detection with AprilTags, and joint planning for Rexarm and MBot.

For this assignment, you will be working in randomly assigned groups. Each group will have access to the code repository on Gitlab. The code is available on Canvas as well, at `Files/A3/a3_code.tar.gz`. In this assignment, you will mainly modify the code in the `armlab-f19` folder inside the repository (or if you download the code from Canvas, you will see this folder after extracting `a3_code.tar.gz`).

Setup

In this lab assignment, you will mainly write Python3 code, and when you run the Python script, you will use the command `python3 YOUR_SCRIPT.py` (replace `YOUR_SCRIPT.py` with the script to run). When you ssh into Raspberry Pi, add a `-X` flag. That will forward the GUI showing up on the Pi to your laptop.

This lab will be done with the Qt GUI interface already provided, which is implemented in `control_station.py`. You will run the script on Raspberry Pi, and forward the GUI to the laptop.

The `control_station` program launches three threads. First the camera is handled in its own thread, `VideoThread`, to capture and display the RGB image and the detected AprilTags from the camera. The state machine runs in its own thread, `LogicThread`, and is running at a fixed rate determined by the sleep time in the thread's `run()` function. All of the messages to the Rexarm are sent using the state machine for both commands and feedback. The third thread, `DisplayThread`, displays and updates all of the read-outs from the Rexarm and State Machine.

When you launch the program, you can put the arm in the `manual` state by clicking the manual check box above the grayed out sliders. This will switch states from idle to manual and let you control the individual motors with the sliders.

Now that you can control the arm manually with sliders, and you will notice that there are physical limitations for each joint. If you command a value greater than which is physically achievable, the servo will drive up against the physical obstacle and can therefore be damaged. Use the sliders on the GUI to change the joint angles gradually, and find the joint limits. Write a clamp function in `rexarm.py` to constrain the joint angles of the arm, before writing other functions that directly command the joint angle to a specific value. If you want to command the joint angles, always clamp them before commanding.

The Rexarm on the Mbot has 5 motors (4 revolute joints plus a gripper), and they are Dynamixel motors model XL-320. There are other Dynamixel motors available in the lab. You can use them for the final project, and 3D print the links.

MX-28	AX-12	XL-320
Data Sheet	Data Sheet	Data Sheet
Voltage: 12V	Voltage: 12V	Voltage: 7.5V
Stall Torque: 2.5 N·m	Stall Torque: 1.5 N·m	Stall Torque: 0.39 N·m
Max speed: 55 RPM	Max speed: 59 RPM	Max speed: 114 RPM
Encoder: Magnetic (360°)	Encoder: Resistive (300°)	Encoder: Resistive (300°)
Resolution: 0.088°	Resolution: 0.29°	Resolution: 0.29°

Fig. 1. Dynamixel motors available in the lab.

Task I: Forward Kinematics

In this task, you will calculate the position of each link and the end effector (the gripper), based on the joint angle of each motor. You will assume that the base of Rexarm is the origin of the coordinate system, and the forward direction of the Mbot is the positive x-axis direction, and the upward direction is the positive z-axis direction. You will be dealing with another coordinate system (camera frame) in Task III, and these two coordinate systems are different.

Measure the length of each link using rulers, then hard code them in `rexarm.py`. You will use Denavit-Hartenberg (DH) convention and fill in the DH table in `rexarm.py`. Implement the linear algebra that calculate the position of the end effector. Display the position of the end effector in the Qt GUI by running `control_station.py`.

Set the torque of each motor to 0, so that you can move the Rexarm with your hand, and see whether the positions of each link and the end effector are reasonable.

Submit: Include a DH table for the Rexarm in your report.

Task II: Inverse Kinematics

Modify the code in `rexarm.py`. Given a desired pose of the end effector, use trigonometry and geometry relationship between the links to calculate the joint angles.

Always use elbow-up configuration for the arm. Implement different reaching schemes (normal reach, reaching high, and reaching far) discussed in the lecture, or come up with your own schemes. If there are no feasible joint angles that make the end effector reach the desired pose, the program should return `None`.

After you implement the inverse kinematics, use your forward kinematics to verify the implementation, and check if any link has collision with the Rexarm itself, or with the Mbot, or with the ground. If the calculated joint angles lead to a collision, return `None`.

Submit:

1) Report all the formulas you use to calculate the inverse kinematics.
2) Include schematic diagrams of the arm that label the symbols of the angles that you used in the formulas.

Task III: Camera Calibration and Object Localization with AprilTags

For this task, you will implement the algorithms that get Rexarm frame coordinates of AprilTags [1] from the camera. To do so, you will need to calculate the intrinsic and extrinsic matrices of the camera, by performing camera calibration [2].

You can find the intrinsic matrix for the camera by using the provided camera calibration Python program (`util/camera_cal.py`) and provided checkerboards. This program will also find lens distortion parameters which you can use to transform the images to have less distortion, though this is not necessary as the camera image has relatively low distortion.

After you get the intrinsic matrix, the AprilTag detection function can calculate the pose of each tag in the camera frame accurately. Modify the code in `VideoThread` of `control_station.py` to use your calculated intrinsic matrix.

You will be provided with 1-inch-by-1-inch cubic blocks, with AprilTags attached to 4 of its faces.

However, since the camera frame of reference is different from Rexarm's frame of reference, you will need to calculate the extrinsic matrix to convert the pose of AprilTags in the camera frame to Rexarm frame.

Use `SolvePnP` from OpenCV library to find the extrinsics [3]. The input to this function is an array of n 3D points in the world frame, and their corresponding n 2D points in the pixel frame. The pixel frame is an 2D image of the 3D camera frame. The function will return the extrinsics that transform the pose from the points in the 3D world frame to the 3D camera frame. The template code is in `state_machine.py`. You can click on the `Calibrate` button on the GUI to perform extrinsics calculation.

Use the found extrinsics, implement the function that converts the pose of AprilTags from the camera frame, to the Rexarm frame. Write a function that infers the center of the block in the Rexarm's frame based on the pose of the AprilTag attached to it.

Submit:

1) Report the intrinsic matrix of the camera.
2) Report the number of points and their coordinates in Rexarm frame that you used to calculate the extrinsics. Discuss your strategy to choose the points.

Task IV: Color Discrimination

In this task, you will identify the color of each block. In the challenge part, we will distinguish trash blocks from regular ones based on their colors.

As discussed in the lecture, we can use RGB or HSV color space to represent colors. Modify `DisplayThread` in `control_station.py` so that when the cursor hovers over the real time camera image on the Qt GUI, the GUI shows up the RBG and HSV value of the pixel on which the cursor is. That will give you a sense what's the RGB/HSV range of each color is.

You will be provided with 1-by-1-by-1-inch cubic blocks, with AprilTags attached to 4 of its faces, and you will be able to see the color on other 2 faces. The two colored faces will be two opposite faces. Implement function that identify the color of the block. The available colored blocks are red, blue, yellow, orange, purple and green.

Submit:

1) Observe the RBG/HSV values when the lighting condition changes. What do you find? Report which color space you use to discriminate the color, and discuss the reason.
2) For each colored block, include the range of values of the color space that you choose.

Task V: Path Planning and State Machine for the Rexarm

In this task, you will implement a simple path planner and state machine for the Rexarm to pick up and drop off the blocks. The blocks will be in the reach of the arm, and in the range such that the camera is able to detect the AprilTags.

Given the desired pose of the end effector, find the desired joint angles of the Rexarm using inverse kinematics. Then given the initial joint angles of the arm, plan a collision-free path in the configuration space. There are sophisticated path planning algorithms such as RRT, but they are an overkill for simple tasks. If you move one joint of the arm at a time, it is easy to find a collision-free path. Insert additional waypoints between the initial and desired pose to implement this function.

If you set the position of the end effector to be exactly at the center of the block, you may find that it is hard to pick it up. It will be easier if you shift desired end effector pose slightly to the left/right based on the design of the gripper. You may also find that the end effector may not be able to reach the desired position exactly, because the shoulder joint doesn't have enough torque to hold the arm up.

Submit:

1) Discuss your heuristics to plan a collision free path for the Rexarm.
2) Discuss any difficulty that you meet when picking up or dropping off the block. Discuss your heuristics that shift the pose of the end effector.

3) Write a state machine that picks up a block in the view, and puts it back to the original position. The block should be put at an arbitrary position in the reach of the arm, and the robot should use camera to locate it. Take a video of it and provide the link to the video in your report.

Task VI: Mobile Manipulation

In this task, you will reuse the code in A1 and A2, combine them with the code in this assignment, to realize the mobile manipulation functionality. You will tune the motion controller and SLAM algorithm, and modify the path planning algorithm for MBot. Notice that you may need to convert the coordinates from Rexarm's frame to MBot's frame when doing the path planning.

The program to control the Mbot is mainly in C++, but the code for the Rexarm and camera is in Python. You will use lcm messages to let Python program communicate with C++ program.

You will come up with your own unique and creative design of state machines to finish the tasks.

Implement the state machine that identifies the color of a block. Since none of the two colored faces of a block may be facing the camera, the robot may need to rotate the block until it can see a colored face. You can also program the robot to wander around the block, until it can see one colored face.

Implement a state machine that picks up a block jammed in a corner formed by two walls. The robot will first need to poke the block out of the corner before picking up.

Implement a state machine that picks up a 3-by-1-by-1 inch block which can only be grasped from certain direction. The robot will need to poke and rotate the block such that the block becomes easy to pick up. Or the robot will need to go around the block, and find the position it feels most comfortable to pick the block up.

Submit:

1) Discuss the difficulties you met integrating the whole system together.
2) Record a video that the robot picks up a block jammed in the corner.
3) Record a video that the robot picks up a long block that is initially impossible to grasp.
4) Provide links to the videos in the report.

Challenge: Cleaning Up an Environment

We will be holding a challenge in the classroom on October 28 during the regular lecture session.

Your cleaning robot will explore a 1.8m-by-1.8m environment constructed by the course staff. It will need to pick up blocks well-separated in the environment, and send them to trash bin or supply area. Both trash bin and supply area will be marked with AprilTags. The blue trash bin is about 5 inches tall. The supply area is a red letter size paper on the ground. The red, green and orange blocks are considered as trash, and they should be sent to the trash bin. The rest of them should be sent to the supply area.

Initially, the robot will be put roughly at (0.9 m, 0.4 m) inside the environment (the bottom left corner of the environment is treated as the origin (0.0 m, 0.0 m)), facing the supply area.

There will be in total 6 blocks in the environment, and they will have different attributes and points.

- One block will be close to the initial pose of the robot, and it will be in the view of camera if the robot rotates itself.
- At least one block will be initially outside the view, but inside the mapped region. The robot will wander in the environment and find the block.
- One block will be blocked by an obstacle initially, inside the unexplored region of the map, and the robot will need to explore the environment to find it.
- One long block that the robot will need to go around the block and find the appropriate position to pick it up.
- One block close to the wall.
- One block jammed in the corner and the robot will need to poke it, move it out of the corner and then pick it up.

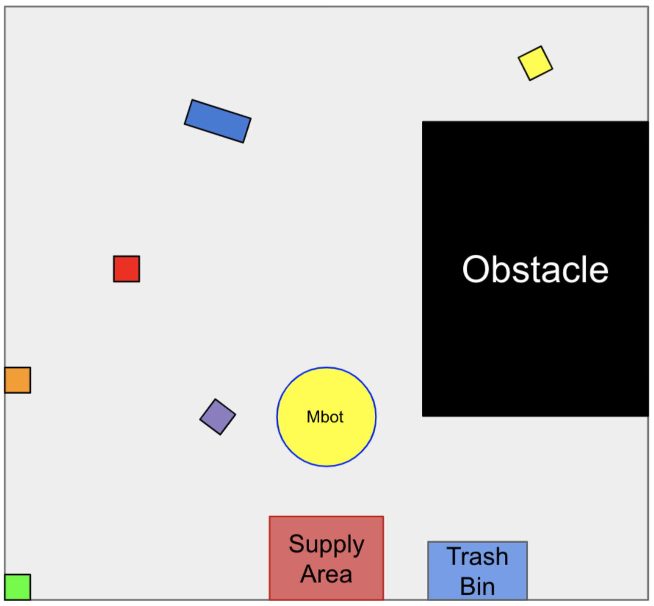

Fig. 2. Arena of the challenge. The position of the supply area, trash bin and the obstacle will be similar to what is shown in the figure. The blocks are well-separated in the environment. The supply area and the trash bin will have AprilTags with id 7 attached to them. The AprilTags on the blocks will have id 1 through 6, and they will have the same id if they are on the same block. The long block will have AprilTags with id 1 on it.

Each group will have two attempts. Each attempt can last at most 15 minutes.

Submit: Discuss your strategy to pick up the six blocks in the environment. Discuss how you implement the state machine and path planner for the blocks with special attributes.

Report: Mobile Manipulation

Each group should produce a report on your project, formatted using LaTeXin IEEE two-column conference format. This style can be found in the ieeeconf package.

Produce the needed figures, and then include them into the report, which is submitted as a PDF file. Each figure should include a detailed caption, explaining the method, conventions, and significance of the information in the figure. The main body of text should refer to each figure, but should not duplicate the caption explanation. (Think of the figure and its caption as a module being used by the main body to communicate the message of the paper.)

Compress all the code into a tar.gz file, `team0X_a3.tar.gz`, where `0X` is your team number, and submit it together with the report. For each team, one submission is sufficient.

Certification [required for credit, 0 points]

Print or write the following on a sheet of paper:

"I participated and contributed to team discussions, and I attest to the integrity of the report."

Each team member should sign the paper (physically, not digitally), and a photo of the paper should be included with your report. By doing so, you are asserting that you were materially involved in the team work, and that you certify that every solution complies with the collaboration policy of this course. In the event of a violation of the course's guidelines, your certification will be forwarded to the honor council.

If the statement requires qualification (i.e., it's not quite true), please describe the issue and the steps you took to mitigate it. The staff will use their discretion based on the specific circumstances you identify, and we are happy to discuss potential problems prior to the report due date.

If your team works together, as required, this will be painless! :)

References

[1] AprilTags for Python3
`https://github.com/duckietown/apriltags3-py`

[2] OpenCV Camera Calibration
`https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`

[3] OpenCV SolvePnP.
`https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnp`