EECS 106B Lab 3: Constrained Control with Turtlebots [*]

Due date: Monday, March 19th at 11:59pm

Goal

Constrained Control with Turtlebots

You will develop controllers pictured in Fig. 1 using motion contstraints. The purpose of this lab is to get hands-on experience in controlling a robot subject to constraints such as

1. Onstacle Avoidance
2. Non-holonomic dynamics (e.g. cars)

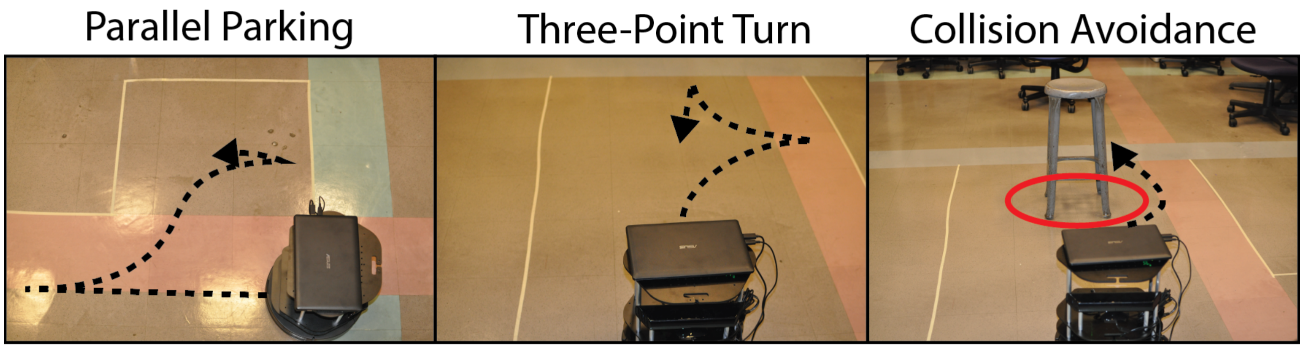

Figure 1: The three constrained trajectories you will follow in the lab.

Contents

[*]Developed by Jeff Mahler, Spring 2017. Expanded and edited by Chris Correa and Valmik Prabhu, Spring 2018

1 Theory

For this lab, you'll be controlling a turtlebot by modeling it with a unicycle model.

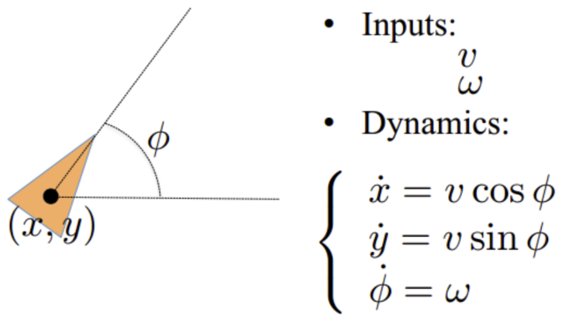

Figure 2: Dynamics of the Unicycle Model

This paper Details several control methods for a unicycle type robot.

For this lab, you will be implementing a velocity controller to follow several paths. We recommend using Controller A for Tracking from the above paper, but you may use any controller you see fit (including one not mentioned in the paper).

Your state space should be the position and angle of the turtlebot, $q = (x, y, \theta)$, and you will control $\dot{q}$. This means that your controller should look like

$$\dot{q} = Aq + Bu \tag{1}$$

Under this model, the non-holonomic constraint can be written as $w \cdot \dot{q} = 0$ where $w = (\sin(\theta), -\cos(\theta), 0)$.

As a review, this means that your velocity controller will take in the robot state (x, y, θ), and potentially its current state velocity, and output a twist u, where

$$u = \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2}$$

Where v is the desired velocity and ω the desired angular velocity.

2 Logistics and Lab Safety

We expect that all students in this class are mature and experienced at working with hardware, so we give you much more leeway than in EECS 106A. Please live up to our expectations.

2.1 Groups, Robots and Accounts

Remember that groups must be composed of 2-3 students, at least one of whom must have completed EECS 106A. Robots should be reserved on the robot calendar here. The rules are

1. **Groups with a reservation have priority.** You can go in and use (or continue using) the hardware whenever you like if no one has a reservation. However, you must pass along the hardware once the next reservation starts. Please be respectful and try to plan ahead; it's not cool to take the first twenty minutes of another group's section winding down.

2. **Do not reserve more than two hours at a time.** This is shared hardware and we want to ensure that all groups have enough time to complete the lab.

3. **One computer per group (if needed)** We only have ten lab computers, and have around thirty students. If more than ten students want to use the computers, please try to limit yourself to one computer per group. In addition, groups with robot reservations have priority on the computers next to their respective robots. Groups can accomplish working in parallel by using personal computers, file-sharing software like git, remoting into the lab computers, and/or using simulators like Gazebo or RViz.

2.2 Safety

Remember to follow the lab safety rules:

1. **Never work on hardware alone.** Robots are potentially dangerous and very expensive, so you should always have someone on hand to help if something goes wrong. This person should be another 106B student, though in cases when group schedules are highly incompatible, you may talk to a TA to arrange an alternate solution, such as bringing a friend (note that you would be entirely responsibe for this person's conduct).
2. **Do not leave the room while the robot is running.** Running robots must be under constant observation.
3. **Always operate the robot with the E-Stop within reach.** If Baxter is going to hit a person, the wall, or a table, press the E-Stop.
4. **Power off the robot when leaving.** Unless you're trading off with another group, the robots should be powered down for safety. To power on/off the robot, press the white button on the back of the robot.
5. **Terminate all processes before logging off.** Leaving processes running can disrupt other students, is highly inconsiderate, and is difficult to debug. Instead of logging off, you should type

   ```
   killall -u <username>
   ```

   into your terminal, where `<username>` is your instructional account username (`ee106b-xyz`). You can also use `ps -ef | grep ros` to check your currently running processes.
6. **Do not modify robots without consulting lab staff.** Last semester we had problems with students losing gripper pieces and messing with turtlebots. This is inconsiderate and makes the TAs' lives much more difficult.
7. **Tell the course staff if you break something.** This is an instructional lab, and we expect a certain amount of wear and tear to occur. However, if we don't know something is broken, we cannot fix it.

2.3 Questions and Help

If you experience software-related errors, please perform the following:

1. Document exactly what you did leading up to the error (commands, terminal output, etc.)
2. Post on Piazza with a description of the error, the above materials, and a description of what you did to try and fix it.

Chris and Valmik **Will Not** diagnose any programming errors without the above documentation. However, feel free to ask us theoretical questions on piazza, during office hours, or after lecture or discussion.

3 Project Tasks

You'll be using the Turtlebots in Cory 111.
For this lab, you'll write controllers to follow trajectories using two types of constraints:

1. *Non-holonomic: Unicycle Model* Develop your own velocity controller to perform the following maneuvers using a unicycle model:

   (a) Parallel Parking

   (b) Three Point Turn

   These are pictured in the left half of Fig. 1.

2. *Holonomic: Obstacle Avoidance* Develop a velocity controller to follow a shortest-path trajectory in the presence of an obstacle. The controller is constrained to the set of poses for which the Turtlebot does not collide with the obstacle (aka the Free Configuration Space). We recommend approximating the object by a cirlce of radius r centered at location p relative to the starting point of the Turtlebot (you may want r larger than necessary to give clearance). This is pictured on the right side of Fig. 1. Compare the following control methods:

   (a) Standard controller for a *straight-line* trajectory through the object with an additional term proportional to the distance of the Turtlebot to the obstacle that models a virtual "constraint force". *Note: We are not providing you with any algorithms for this part. You should develop your own solution; be inventive.*

   (b) Precompute the shortest possible trajectory around the obstacle and use a standard controller.

4 Deliverables

To demonstrate that your implementation works, deliver the following:

1. Videos of your implementation working. Provide a link to your video in your report.

2. Provide the code in a zip file, to be uploaded with your writeup.

3. Submit a detailed report with the following:

   (a) Describe your approach and methods in detail, including a formal description of the controllers you used.

   (b) Let us know what of any difficulties and how you overcame them.

   (c) Summarize your results in both words and with figures. In particular, show:

      - Plot the true Turtlebot state (x, y, θ) reached with your controller versus the desired state for each control task.
      - Analyze trajectory tracking performance over different values of the control gains.
      - Describe any constraint violations that occurred during execution (e.g. lateral velocity, collisions)
      - Compare the performance of the two obstacle avoidance controllers (constraint forces vs. pre-computed trajectory) in terms of speed and accuracy.

   (d) Compare your controllers with the equations of motion in equation 6.5 of MLS. How do your controllers take the virtual constraint force into account?

   (e) In practice many obstacles may not be well-approximated by a single circle. How might you extend our model of the circular collision avoidance constraint to improve performance?

   (f) In the Unicycle Model paper, the author provides controllers for both trajectory tracking and path following. What is the difference between these methods? When would you want to use one as opposed to the other?

4. **BONUS:** In addition to your report, write a couple paragraphs on the difficulties you had performing the lab, paying particular attention to how the lab documentation could be improved. This course is evolving quickly, and we're always looking for feedback.

5 Getting Started

We assume that you've already completed all the configuration steps from labs 0, 1, and 2. There are some different steps for lab 3.

5.1 Configuration and Workspace Setup

Note: *we expect all students to already be familiar with configuration procedures, either through EECS 106A, Lab 0, or some other place. Therefore, we won't be providing any explanations in this section. Please consult Lab 0 if you require a refresher*
Type the following commands:

```
cd ~/ros_workspaces
mkdir ~/ros_workspaces/lab3_ws
mkdir ~/ros_workspaces/lab3_ws/src
cd ~/ros_workspaces/lab3_ws/src
catkin_init_workspace
cd ..
catkin_make
source devel/setup.bash
```

Now download `Lab3_Resources.zip` from bcourses and unzip it. Put the `lab3_pkg` package into the `src` folder of `lab3_ws`, and rebuild.

5.2 Working With Robots

For this lab, you'll be working with Turtlebots. There are three turtlebots: Red, Blue, and Yellow. To start, you must power the turtlebot base. Once the robot is powered on, check connectivity by running

```
ping <color>.local
```

Where `color` is either `red`, `blue`, or `yellow`. Now, ssh into the Turtlebot from any lab workstation:

```
ssh turtlebot@<color>.local
```

The password is "EE106B18". Then run

```
roslaunch turtlebot_bringup minimal.launch
```

In a new terminal, run

```
hostname -I
```

You should see one IP address of the form `192.168.1.X`. Write this down. Open up your `.bashrc` file. Comment out any existing lines specifying `ROS_IP` or `ROS_MASTER_URI`. Then add the following lines

```
export ROS_MASTER_URI=http://<color>.local:11311
export ROS_IP={YOUR_192_IP_ADDRESS}
```

and save. Make sure to reload the `.bashrc` file before running Turtlebot examples. Note that you may have to change the IP address if you switch workstations, and will have to change the URI if you switch turtlebots.
You can check that the Turtlebot services are up by running the following command from the workstation.

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Then you can control the Turtlebot using the keyboard (the controls will be shown on the screen).

5.3 Starter Code

The starter code is very similar to that of lab 1. It contains four nodes: `main.py`, `paths.py`, `controllers.py`, and `util.py`.

5.3.1 Main

This file contains the main loop of the code. It looks at the position of the turtlebot, calls the controller to find the desired control input, and sends it to the turtlebot. You will have to modify the hyperparameters at the top of the file and define `current_state` depending on the test you're running. You should familiarize yourself with this code first.

5.3.2 Paths

This file contains the target states and velocities for the turtlebot, given some timestep s. You should start writing code here. The linear path should be straightforward. The arc path should be a circular path, but support segments of a circular path. Both paths should also support going backwards along the path, since backward motion is required for both three point turns and parallel parking. `ChainPath` is similar to `MultiplePaths` from lab 1. You can create the full paths for parallel parking and three point turning at the bottom of the file by utilizing `ChainPath`. You can visualize the path you created by running

```
python src/paths.py
```

5.3.3 Controllers

Here you should write your controllers. They should take in a path object and the current state of the robot and compute a twist message as a velocty command to send to the robot. Note that the function has additional input arguments for your force-based obstacle implementation.

5.3.4 Utils

You will find useful functions here, specifically the `twist_from_tf` and `rigid` methods.

6 Scoring

For each task listed in Section 3, you'll be graded on a 0-5 score as explained in Table 1.

Table 1: Grading Rubric for Lab 3

Score	**Meaning**
0	Did not attempt
1	Attempted but missing more than three deliverables
2	Complete with major error or missing two or three deliverables
3	Complete with moderate error or missing one deliverable
4	Complete with minor errors
5	All tasks completed satisfactorily with deliverables

In addition, you'll earn 5 points for submitting your code. The bonus writeup will be worth the same amount of points as one of the tasks, and will be graded on the same scoring system. Finally, the regrasping and orienting bonus task will be worth 10 points (the five point scale will be doubled). See Table 2 for more info:

Table 2: Point Allocation for Lab 3

Section	**Points**
Video	5
Code	5
Methods	5
Parallel Parking	5
Three Point Turn	5
Trajectory-Based Obstacle Avoidance	5
Force-Based Obstacle Avoidance	10
Discussion of Results	5
Theoretical Questions	5
Bonus Difficulties section	5*

Summing all this up, this mini-project will be out of 50 points, with an additional 5 points possible.

7 Submission

You'll submit your writeup(s) on Gradescope and your code through bCourses as a single .zip file. Your code will be checked for plagiarism, so please submit your own work. Only one submission is needed per group.