Homework 2 (3% over the final grade)

Assigned: August 29, 2024

Due: September 3, 2024

Homework 2 will introduce you to the tools we will use in this course. Most of the commands are specified for you below.

Part 1

Ensure your laptop is setup correctly following instructions in Project 1

Part 2

Setup your git repo. The instructions below use github. This repo will be used throughout the semester

- Create an account (if you don't have one already) on https://github.com
- For the class repo we will use a template so that you will have the same folder structure. Go to: csce274-base (https://github.com/18r441m/csce274-base) and click the green "Use This Temple" button.
- Name your repo using only lowercase letters. Make sure to set it to PRIVATE. This should be free. If not, make sure you used your email.sc.edu email.

Part 3

All commands below are intended to be run from within your Linux install.

Customize Dockerfile and Run docker container

- Clone the git repo from the previous part on your computer by running:

```
git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```

- Use "cd your-git-repo-directory" to change the directory to your repo.
- Using the text editor of your choice, edit "Dockerfile". If in doubt, type:

```
gedit Dockerfile
```

- Change lines 2-4 to your repo name, description, and contact info.
- You should commit these changes before we go further. Run:

git status
And you should see this Dockerfile has changed.

- To commit the changes, run:

git commit -am "Updated Dockerfile"

The -a flag tells git to commit all changes (this can sometimes commit more than you want, so be careful when using it in the future). The -m flag let's you put a commit message on the command line. You can change the part in quotes however you like.

- Now you need to push this to your repo:

git push origin master

- Build the docker image contained in this repo using dts tools:

dts devel build -- pull

You have now created a docker image called duckietown/<your_repo_name> (). You only need to run this build command when something in the dockerfile changes. You will likely be able to use this same image for your first several assignments.

For those who know docker already: this is essentially equivalent to running docker build, with a few defaults set by Duckietown to make it easier.

- Confirm your image was made by running:

docker image ls

Look for something like duckietown/<your_repo_name>

- Start your docker container using dts:

dts devel run -M -- --entrypoint=/bin/bash

Let's dissect this command a bit:
The command "dts devel run" will start a Duckietown-based container (like what is in the class repo you used as a template) with a configuration that lets it run like Docker images will run on your robot. This saves us from writing a very long docker run command with many options. Proper documentation on this tool is still forthcoming, but some information can be gleaned from looking over this tutorial if you are interested in deeper understanding:
https://docs.duckietown.org/daffy/opmanual_developer/opmanual_developer/out/index.html This tutorial is NOT required for HW2.

The -M option mounts the current project to the working directory of the docker container. This allows us to write code outside of the container and run it inside, which can be handy depending on which text editor you prefer. We won't use this in HW2, but it will be our method for future homework.

Everything after -- is sent directly to the underlying docker run command. In this case, we just want to use a normal terminal, so we override the default entrypoint (script that runs when the container starts). You can experiment with scripts in the "launcher" directory of your repo if you want to avoid overriding the entrypoint each time.

If you run "ls" inside this container you should see the contents of your repo here. These files are shared with your computer right now, so when you make changes on your computer they will also be changed in docker and vice versa.

- We need to build the current ROS environment inside this container. This is largely done for us in Duckietown containers, but it is always good practice to double check. Run:

```
cd /code/catkin_ws/
source devel/setup.bash
catkin build
source devel/setup.bash
```

These commands allow you to use the correct environment for ROS and build everything in that environment. You will have to run these commands every time you change code or create new packages. For this assignment, once is enough.

Part 4

Start a ROS node

- First, let's start roscore:

```
roscore
```

This will run so that you cannot do anything else in this terminal until it is time to end roscore.

- So, let's start another terminal and connect it to your docker container. Run:

```
docker ps
```

And find the name of your container. It should be the only one running. Record this name. For example, the name of the container could be: "dts-csce274-template"

- Then run:

```
docker exec -it <CONTAINER_NAME> bash
```

Replacing <CONTAINER_NAME> with the name you found

You now have a second connection to your docker container. You need to source your environment again, because this is now a new shell with a new environment:

```
source /code/catkin_ws/devel/setup.bash
```

- Run the ROS node used in HW2:

```
rosrun homework2 homework2.py
```

You won't see any output from this command unless there is an error.

- Open a third terminal and run the docker exec command from above again.
  In the third terminal connected to your docker container, source your environment (5c) again. Remember you will have to do this every time you open a new terminal connection.

- Then run:

```
rosnode list
```

You should see "/homework2" listed here.

- This seems like a large number of terminals just to run a node and see what's going on. Good news! There is a better way! Press ctrl-c in each of these three terminals to end their current process and move on to the next step.

Part 5

roslaunch and rostopics

- From any of the terminals you used in the last part, run:

roslaunch homework2 homework2.launch
This will print some debug text to the screen and then stop printing information.

- From either other terminal from part 4, run:

rosnode list
This shows all of the nodes that are currently running. You should see the same node you saw before.

- In any terminal you set up, run:

rostopic list
This shows all active topics. Two topics, “/rosout” and “/rosout_agg” are almost always shown and we can ignore them for this assignment. We will focus on the two topics that are part of homework2.

Let’s figure out what is going on with this ROS node so we can use it. Use the rostopic info tool for both homework2 topics. Record the publisher and subscriber for each one as well as the message type.

rostopic info <topic>

One topic will say that is has a publisher but no subscriber, the other will be the opposite. This means that the ROS node homework2 is listening (subscribing) to one topic and talking (publishing) over the other. We can listen to the output of the ROS node using a command line tool. This is very useful for debugging. Run this command:

rostopic echo <topic>
For the topic that you think is being sent by the homework2 node. Leave this command running.

- You shouldn’t see any output right now because the node is not publishing anything. Usually, that means the node is waiting for some input. Let’s give it that input. You will need the name of the topic you decided the node is listening to as well as the type of the message. Run this in another terminal:

rostopic pub -1 <topic> <message_type> <value>
HINT: the value is just a number in this case. Pass any number you like.

NOTE: that is a “dash one” (1) after the command. Some fonts are clearer than others. It makes pub send the message exactly once and exit.

- You should see some output on the terminal running rostopic echo. Send a few different values and see what happens. Record the inputs and outputs. What do you think this node is doing?

Part 6

Github tags

You can end all of the above processes (ctrl-c) and quit any docker containers (type “exit” in their terminal windows). Now we will push (save) the code that we just ran. For this assignment you are submitting the same code you were given, but in future assignments you will write new code.

- In a terminal, change to your repo directory and run:

git status
Unless you made changes to the files, this should say “nothing to commit”

Even though there is nothing to commit, I ask that you tag each of your submissions so I know which commit to grade. We are going to practice that now. Tags should generally be HW# for homework and P# for projects.

- To tag this assignment, run the following in your repo directory:

git tag hw2
git push origin hw2
Remember to do this for each assignment you turn in!

- Go to the repo releases page and download the zip of the tag your just created and upload to Blackboard along with the content from the next step.

Complete the instructions in part 7 to submit to Blackboard

Part 7

Blackboard submission

- Submit a PDF to Blackboard with:
    - The name of your docker container used?
    - Which topic was being published by the homework2 node?
    - Which topic was being subscribed to by the homework2 node?
    - What message type did the topics have?
    - What is homework2 doing? What were the inputs/outputs you used?
    - pdf format instructions
        - Header with the code of the class, the semester and year, the homework number, and your name.
          e.g., CSCE 274 Fall 2024 – Homework 2 – Ioannis Rekleitis
        - Your answers, clearly identifying the answered questions.
        - The name of the file should be in the following format:

csce274_fall2024_<hw#>_<last_name>.pdf
e.g., csce274_fall2024_hw2_rekleitis.pdf

- Sumbit the zip file from the previous step to Blackboard