CSCE 439/839: Robotics: Algorithms and Applications, Spring 2022
Homework 1

Started: Weds, Jan 19th
Checkoff 1: Fri, Jan 28th in class
Due: Thurs, Feb 3rd, midnight

Instructions: This homework is an individual assignment, collaboration is not allowed. If you discuss any problems with others, please note this on the assignment as described in the syllabus. Also note any materials outside of lecture notes, course textbooks, and datasheets that you used. Show your work and describe your reasoning to get partial credit if your solution is incorrect.

You should also make sure that you properly label and **describe** any figures or plots that you include in your writeup. You will not receive full credit if you do not explain these. In addition, you should refer to your code where needed to answer the questions (e.g. say "See file mynode/launch/test.launch for this problem, which ...").

You must turn in a pdf of your assignment on Canvas. Make sure to answer questions in complete sentences and explain answers as needed. **You must also turn in your code for all parts of this problem on Canvas.** Failing to electronically turn in your code will result in a 10 point penalty on this assignment. Points may also be deducted for coding errors, poor style, or poor commenting.

If you use any code from an online or other source you must cite the source in the comments. Otherwise it is considered plagiarism, which we check for!!

Name:

Problem 1. *(5 pts)*[1] *(To be completed at end of assignment) Approximately how much time did the total assignment take? Which sub-problem took longest and how much time did it take? Are there any questions that need clarification?*

Problem 2. *(10 pts) Installing Docker. Download and install Docker from* `https://www.docker.com/get-started`*. Launch Docker and then start the container for the initial tutorial with the command* `docker run -d -p 80:80 docker/getting-started` *as described in tutorials* `https://docs.docker.com/get-started/`*. Complete at least the getting started tutorial* Part 1: Getting started *through* Part 6: Use bind mounts.
Checkoff: This checkoff is due by Friday, Jan 28th in class. Bring your computer[2] to class to get this checked off by showing how you have been able to create a persistent database using Docker as outlined in the tutorials. Note that the next checkoff is also due the same day!

[1]Each HW counts equally in your overall grade, even if homeworks have different point totals. This one is out of 90 points for 439 and 90+20 points for 839.
[2]If you do not have a laptop, please talk to me now.

Problem 3. *(10 pts) Installing ROS and testing serial latency*[3]*. Note, you will need an Arduino for part of this problem. The Arduino will be distributed during class in the 2nd week.*

On the course website there is a zip file that includes a Docker file that contains ROS and additional code that is required to forward serial data from your computer into the Ubuntu system running in the Docker container. Then follow the following steps (mostly for Mac and Windows, but similar steps can be followed for most Linux installs, if you have questions, start a thread in Canvas):

- *Unzip the code from the course website and using a terminal go to the directory where you unzipped this.*
- *Build the Docker container by running this command from inside the 439-839 directory:*

  ```
  docker build -t ros .
  ```

  This will build the container based on the specifications in the Dockerfile. It may take a while as it needs to download a few gig and build a whole bunch of stuff. While you wait, you can continue the following steps.
- *On your computer install python3 from `python.org`. On Windows, select "advanced options" (or possibly "Add to PATH") during install and make sure to check "Add python to environment variables" to make sure that you can run python from the terminal. On Macs, select "Customize" and make sure that "UNIX command-line tools" is selected. (Note, a reboot may be required after installing.)*
- *From the terminal on your computer, install required libraries with pip:*

  ```
  pip install serial serial-tool
  ```

  Note: If you have problems where the python code later on doesn't run, you may need to `pip uninstall serial serial-tool` *and then install them again. Sometimes python gets confused depending on what you previously had installed.*

  Note 2: It may be that you need to use pip3 in addition to pip depending on your install.
- *While you continue to wait for the docker image to build, download the Arduino IDE from `https://www.arduino.cc/en/software` if you do not already have it. Look at and load the sketch found in the "latency-test-arduino" directory in the zip file. We will use this shortly to test the latency of serial communications with our Docker container.*
- *Now hopefully the docker build is done (if not wait until it is for this next step). Now run the docker image from inside the 439-839 directory. It is critical to run it within this directory show that the "shared" directory is properly shared with the container.* **Remember** *only files you put in the shared directory when running will be preserved when the container is closed! So make sure you only put files you care about in that directory! To run the container do:*

  ```
  docker run --rm -v "/$(PWD)/shared:/home/shared" -p 5901:5901 -it ros
  ```

  If that doesn't work on Windows you might need to specify the path (where `/c/439-839` *may need to be updated based on the current directory):*

  ```
  docker run --rm -v "/c/439-839/shared:/home/shared" -p 5901:5901 -it ros
  ```

  This shares the shared directory and exposes port 5901 to use with a VNC client. It should startup and spit out some messages saying it is trying to create a serial bridge. We will work on that next.
- *On your machine, make sure your Arduino is still connected and then in another terminal go to the directory 439-839 and run:*

  ```
  ./tcp_serial_redirect.py
  ```

[3] You should explore `http://www.ros.org` as there are many more helpful hints and documentation that are not referenced in this homework.

OR on Windows potentially:

```
python ./tcp_serial_redirect.py
```

OR just

```
tcp_serial_redirect.py
```

This will output a list of serial ports that are available, for instance on my machine I get:

```
---------- Serial Ports Available ----------
/dev/cu.usbserial-FT9IJY7Q
 name: cu.usbserial-FT9IJY7Q
 desc: TTL232R-3V3
 hwid: USB VID:PID=0403:6001 SER=FT9IJY7Q LOCATION=2-1.2.2.1
/dev/cu.usbmodem101
 name: cu.usbmodem101
 desc: IOUSBHostDevice
 hwid: USB VID:PID=2341:0043 SER=9523234373335130A111 LOCATION=0-1
--------------------------------------------
```

On my machine `/dev/cu.usbmodem101` *is the Arduino, so next run (replacing the port with whatever your Arduino is on):*

```
 ./tcp_serial_redirect.py /dev/cu.usbmodem101 115200
```

This opens the specified serial/baud and waits for a tcp connection on port 7777. The Docker container should automatically connect to this after a few seconds (you will see a "connected" message) and create the serial device `/dev/tty_serial_bridge`

- *Launch some sort of VNC client (on a mac you can run the application "screen sharing" and on windows you could download* `realvnc.com` *or any other VNC client). Run your VNC client and tell it to connect to* `localhost:5901` *and the password is simply* `password` *(if you want to change the password, which isn't a bad idea, look for where it is defined in the Dockerfile). You now should see a Linux desktop.*

- *Through VNC you can start a new terminal within the Docker container. Change to the* `shared` *directory (*`cd shared`*) and then run:*

  ```
  ./latency_test.py
  ```

  This opens `/dev/tty_serial_bridge` *and tests the latency by connecting to the Arduino. Feel free to explore this script and* `tcp_serial_redirect.py` *which do the magic of forwarding serial communications from Windows/Mac to the Docker container. On my machine I see mostly 2ms latency, sometimes 4 or 5. You can show us what you end up with during the checkoff.*

Checkoff: This checkoff is due by Friday, Jan 28th in class. Bring your computer to class to get this checked off by showing Docker running the latency-test.py script.

Problem 4. *Turtle Sim. Complete section 1.1 the beginner tutorials found at* `http://www.ros.org/wiki/ROS/Tutorials`. *This goes through a "turtle" tutorial that you will need to use to answer the below questions.* **Note:** *There is already a catkin_ ws directory in the Docker container under the shared directory. In addition, the Docker container also has a symbolic link from* `/home/catkin_ws` *into* `shared/catin_ws` *so you that you do not accidentally create things that will be erased when the Docker container closes.*

Important Docker Note: *Remember that only files you put in the shared directory when running the Docker container will exist after you close the container. I would recommend that you use your favorite code editor (emacs, of course) on your native machine, not within the docker container, to edit code in the shared directory. That will help you to make sure that you don't inadvertently code up something and have it disappear when you close the container. I will accept excuses like "I got it all working, but it wasn't in the shared directory so it disappeared" as much as I'll accept excuses like "My hard drive crashed and I didn't have backups" or "My dog ate my homework." Which is to say I won't accept any such excueses.*

a). *(5 pts) Use* `rostopic pub` *from the command line to write "ROS" with your turtle. Attach a picture showing your final "ROS" as drawn in TurtleSim.*

b). *(5 pts) Give all the commands you used to write ROS in TurtleSim.*

c). *(5 pts) Show the* `rqt_plot` *of the x and y position, angle, angular velocity, and linear velocity (so five lines) of your turtle as it writes ROS. Augment the plot to show where drawing of each of the letters occurred.*

Problem 5. *Creating New ROS Nodes*

a). *(10 pts) Now create a new ROS node (see* `http://www.ros.org/wiki/ROS/Tutorials` *for details, you can use either C++ or Python) that publishes messages to write ROS (similar to the manual* `rostopic pub` *from problem 4a). Include a picture of the results. Also, describe how you dealt with the delays between sending messages. Remember that you need to comment the code appropriately for full credit (and turn it in using Canvas).*

b). *(5 pts) Create a launch file that starts both the TurtleSim node and your new node. Include a copy of the launch file in your report.*

c). *(5 pts)* **839 Only:** *Create* **one** *new node that writes "ROS" using three different turtles. Include a picture of the results.*

d). *(5 pts)* **839 Only:** *Create three different nodes (or they could be one node instantiated three different times) that each control a different turtle to write ROS (one turtle per letter). Include a picture of the results. The result should be the same as for the prior question.*

Problem 6. *(An Arduino will be distributed during class in the 2nd week) The Arduino kit has a breadboard with some buttons. In this problem, you will program the Arduino to read the buttons and then send this data to a ROS node you will create to ultimately start moving your turtle.*

a). *(5 pts) Program your Arduino to read the inputs of one or more buttons (you can refer/use some of the referenced libraries in the Arduino software). Make sure to include your code in your Canvas submission and tell me here if you wrote your own code or if you used an existing library (include a link to it in your report if you used an existing library). You should print out the values of the button(s) over the serial port in an easy to read format (e.g. labeled "button1: on" etc.). Include a few lines of this output in your writeup and annotate it to indicate what is going on.*
Checkoff: Show the output of this code to the instructor. There will be an opportunity for this during Lab on Friday, Feb 4th.

Problem 7. *Creating a ROS interface to the Arduino.*

a). *(10 pts) Create a binary protocol that includes at least a start byte (a unique byte at the start of each packet) and a checksum to transmit the button information from the Arduino. Even if you only have or hookup a single button, make sure your protocol supports at least 4 buttons for future expansion. Note that by binary protocol, I mean that you should transmit the byte 0x01 instead of the string 'on' or the number "1". You can use any type of checksum you would like (e.g. just the sum of the bytes, XOR, etc).* **Describe** *your protocol in this report.*

b). *(5 pts) Create a ROS node to process the data sent from the Arduino. You can use the* `read_serial.py` *python script (on the course website). This creates a node that publishes each byte it receives on a topic called* `\rx`*. You can subscribe to this topic to get the data, but you should create a new node (in C++ or python) to do the processing of the data received on this topic. Make your node compatible with the* `turtlesim_node` *ROS node. By this I mean that your node should publish the same topics that can be consumed by the* `turtlesim_node` *node.*

c). *(5 pts) Figure out how to use the button to drive around the turtle in TurtleSim. You can use multiple buttons or just a single button. With a single button you could have the first press start a turn, second press go forward, etc. Include a picture drawn by you while controlling the turtle with your button. Be creative in the picture you draw.*

d). *(5 pts) Now create a launch file to launch all the nodes to drive around the turtle with your Arduino.*

e). *(5 pts) Include a printout of the* `rqt_graph` *of your overall system.* **Remember, that all problems that ask for figures or launch files should also include a description to receive full credit.**

f). *(10 pts)* **839 Only:** *Write additional ROS and Arduino code to enable sending a boolean value on the topic* `ArduinoLED` *that will cause the LED on the Arduino to turn on or off based on the value. This will require modifying your ROS node, sending a serial command (modification of the* `read_serial.py` *code), and processing this command on the Arduino. Make sure to include a header and checksum on the binary packet you create (to allow for future expansion). In addition to submitting the code on Canvas, submit a short video that shows your code in action. Make sure you narrate your video.*

Do not forget to fill in the amount of time you spent on this assignment in Question 1.