BARC System Identification

In last week's lab, you learned about BARC's different sensors, and practiced sending simple open-loop commands to the actuators. The purpose of this lab is to give you more insight into the actuators. Through several experiments, you will identify actuator models mapping your actuation command to a specific vehicle response. Knowledge of these models will be crucial to your future control designs.

IMPORTANT: Immediately before running your experiments on the ground, charge the LiPo balance to full charge using BALANCE CHARGE MODE on the charger settings. Make sure the battery stays ABOVE 7V TOTAL (3.5V PER CELL). When it reaches 7V, recharge it to full with BALANCE MODE if you are using it right away again. If you are done with the BARC, store it around 7.6V with STORAGE CHARGE MODE.

Task 1 Velocity

Recall that the longitudinal dynamics of the BARC can be controlled by sending a proper signal to the electric motor. The BARC's Arduino sends a PWM command to the Electric Speed Control (ESC), which then applies a corresponding voltage to the motor. The motor PWM signal is defined on the range $u_m \in [1000, 2000]$, such that:

$$u_{m,PWM} \in [1000, 2000] \begin{cases} 1000 \leq u_m < 1500 : & \text{braking torque} \\ u_m = 1500 : & \text{no torque} \\ 1500 < u_m \leq 2000 : & \text{traction torque.} \end{cases}$$

Since the electric motor parameters are unknown, we cannot establish an analytic relationship between the actuation input and the actual torque transferred to the vehicle. Instead, you will derive a black-box model from motor PWM to vehicle speed. The model is identified by applying step actuation inputs, measuring the vehicle wheel's angular displacement from the encoders, and thus deriving the vehicle's speed response. By comparing signal-vs-time plots of your actuation PWM commands and the vehicle speed, you will be able to identify a black box model for your longitudinal forward dynamics.

1. Open and read through the `lab4.launch` file in the `labs/launch` folder. The file launches several nodes:
   - The Arduino, which records signals to the ESC and from the encoders.
   - The IMU, which collects information about the vehicle's linear and angular accelerations. This will be used during the later steering model identification tasks.
   - The camera, which you will use during the dynamic steering identification task. Camera recordings can be toggled on by changing the `camera_on` value in the `record_experiment` node to `true`.
   - The `SpeedModelID.py` file. `SpeedModelID.py` sends a series of PWM step inputs to the motor, holding each signal for two seconds before sending the next step input. The actuation signals are as described below:

(a) $u_m = 1580$: The vehicle will start to move slowly.
(b) $u_m = 1620$: The vehicle accelerates.
(c) $u_m = 1600$: The vehicle continues to accelerate, but at a slower pace.
(d) $u_m = 1500$: The vehicle comes to a stop.

Recall from the previous lab that actuation signals of message type `ECU` are sent as a pair (motor, servo). For this model identification, you will need to command your vehicle to move as straight ahead as possible. The servo PWM input $servo\_pwm = 1580$ corresponds to a steering angle $\theta = 0$ on our test vehicle, but you may have to tune this parameter. In Task 3 you will establish a relationship between servo PWM and resulting steering angle - in the meantime, use your best judgment.

- The `record_experiment` node, which automatically uploads your recorded data to the Dator cloud after each experiment. You need to collect your vehicle's encoder data and actuation input during the experiment, and `record_experiment.py` automatically saves this information for you. The second line of the launch file sets an `id` argument for the launch file. This indicates that you will need to provide an `id` value each time you run the launch file. Notice how the value you provide as the `id` parameter is used as the `experiment_name` parameter in the `record_experiment` node.

2. Place your BARC vehicle on the ground, and ensure that there is approximately 20 meters of space in front of it. Designate a team member to move alongside the vehicle during the experiment, to prevent any collisions.

3. First, run the data recording service launch file (keep this running in the background while you record multiple experiments):
`roslaunch data_service service.launch`

4. In another terminal, run your first experiment by launching the edited `lab4.launch` file:
`roslaunch labs lab4.launch id:="your_experiment_name"`
where you should replace `your_experiment_name` with your chosen experiment name.
For example:
`roslaunch labs lab4.launch id:="barc_2_accelerationID_1"`
You will use `CTRL-C` to end the experiment when the car has returned to a full stop at the end of the test. You will run this experiment five times, so you may want to use an iteration number as part of your experiment name.

5. Repeat this velocity ID experiment a total of five times, recording the same data during each trial.
Make sure to specify different experiment names each time you call the launch file, so your previous trial's data do not get overwritten.

6. Now you need to extract the data from the rosbags into matlab. Download this **bag_to_csv.py file (LINK HERE)** into the ~/rosbag folder on the BARC (or whatever folder contains your recorded rosbags from the experiments). Run the command `python bag_to_csv.py` from that directory in the terminal. This should generate a new folder called CSV_Files that contains CSV files of parsed data from each experiment. Copy this folder to your computer that is running MATLAB.

7. Once you have the new CSV files folder on your MATLAB computer, download **CSVtoMAT.m file (LINK HERE)** to that new folder. Running that m file should generate a new folder called matFolder which contains mat files of the data in a 1x15 cell array named sig. Each cell contains a time series which you can use for the next calculations. Table 1 on the following page lists the corresponding cell data.

1	roll
2	pitch
3	yaw
4	angular_velocity_x
5	angular_velocity_y
6	angular_velocity_z
7	linear_acceleration_x
8	linear_acceleration_y
9	linear_acceleration_z
10	motor_pwm
11	servo_pwm
12	encoder_FL
13	encoder_FR
14	encoder_BL
15	encoder_BR

Table 1: 'sig' Cell Timeseries

8. Fit a first-order model, (i.e. vehicle speed = f(motor pwm)), to your recorded data:

   (a) Import the data from your five experiments into Matlab. You will mainly be using the actuation input and vehicle encoder data, so you may want to extract those signals out of the struct after importing.

   (b) The encoders provide information about the angular displacement of the vehicle's wheels (**8 counts per revolution**). You can estimate its derivative and calculate the BARC's longitudinal velocity from your recorded encoder data. **You may notice that some of the encoder values are inconsistent and higher/lower.** This is due varying lighting conditions, encoder mounting position and movement. Find which encoders produce reasonable values and use those for your velocity estimation.

   (c) You will use the Least Squares method in order to fit a transfer function model to your collected data. The data may be noisy so take a look at the "smooth" function in MATLAB. Assume that the first-order transfer function is written is the following format: $\frac{V(s)}{U(s)} = \frac{b}{s-a}, a, b \in \mathbb{R}$. This corresponds to a dynamics equation in the form $\dot{v}(t) = av(t) + bu(t)$, where v is your longitudinal velocity and u your PWM input. Concatenate your collected experiment data as:

$$\underbrace{\begin{bmatrix} \dot{v}^1(t_0) \\ \dot{v}^1(t_1) \\ \vdots \\ \dot{v}^1(t_n) \\ \dot{v}^2(t_0) \\ \vdots \\ \dot{v}^2(t_n) \\ \vdots \\ \dot{v}^5(t_0) \\ \vdots \\ \dot{v}^5(t_n) \end{bmatrix}}_{\text{vdot}} = \underbrace{\begin{bmatrix} v^1(t_0), & u^1(t_0) \\ v^1(t_1) & u^1(t_1) \\ \vdots & \\ v^1(t_n) & u^1(t_n) \\ v^2(t_0) & u^2(t_0) \\ \vdots & \\ v^2(t_n) & u^2(t_n) \\ \vdots & \\ v^5(t_0) & u^5(t_0) \\ \vdots & \\ v^5(t_n) & u^5(t_n) \end{bmatrix}}_{\text{v}} \begin{bmatrix} a \\ b \end{bmatrix},$$

where $v^i(t)$ represent the longitudinal velocity at time t during experiment i, $u^i(t)$ is the motor pwm signal, and $\dot{v}^i(t)$ is the acceleration. In Matlab, the Least Squares problem can easily be solved with the backlash operator: $\begin{bmatrix} a \\ b \end{bmatrix}$ = `v\vdot`.

(d) *****Deliverable**: Find the transfer function in the frequency domain, $G_{pwm \to v}(s)$.

(e) *****Deliverable:** Submit two plots:

- A plot of motor actuation PWM vs. time data
- A plot comparing your first-order model and your five recorded velocity vs. time datasets.

Task 2 Braking

We can undertake a similar experiment to derive a black box model of the BARC's speed response to braking commands. Again, we will derive a transfer function from motor PWM to vehicle speed.

1. Append the `lab4.launch` file to launch a node that automatically calls `BrakingModelID.py`, rather than `SpeedModelID.py`, when the launch file is run. `BrakingModelID.py` sends a series of PWM step inputs to the motor, first accelerating and then applying a braking torque. The unedited file applies an actuation of $u_m = 1620$ for five seconds before braking with $u_{m,brake} = 1465$ for seven seconds.

2. As in Task 1, place your BARC vehicle on the ground, ensure that there is approximately 20 meters of free space behind it (since you will now direct the vehicle backwards). Designate a team member to move alongside the vehicle during the experiment, to prevent any collisions.

3. Begin the experiment by running the `lab4.launch` file:
`roslaunch labs lab4.launch id:="your_experiment_name"`
where you should replace `your_experiment_name` with your chosen experiment name.

Make sure to specify different experiment names each time you call the launch file, so your previous trial's data do not get overwritten.

When the vehicle has returned to standstill at the end of each experiment, end the data collection by typing `CTRL-C`.

4. Repeat this experiment five times each for $u_{m,brake} = 1465$, $u_{m,brake} = 1450$, and $u_{m,brake} = 1440$, making sure to change your experiment name for each trial.

5. Fit a first-order model to your vehicle speed data

   (a) Import the input and vehicle speed data from your five experiments into Matlab.

   (b) *****Deliverable:** Find the first-order transfer function from braking PWM to longitudinal velocity, $G_{pwm \to v}(s)$. You can perform another least squares analysis, or use Matlab's `tfest` to estimate the transfer function in the frequency domain.

   (c) *****Deliverable:** Submit two plots:

   - A plot of motor actuation PWM vs. time data.
   - A figure with three subplots, where each subplots compares your five velocity vs. time datasets from a particular $u_{m,brake}$ and your first-order model.

Task 3 Steering - Part 1

Recall that the BARC's steering is controlled by a servo. As you discovered in last week's lab, the servo PWM signal is defined approximately such that:

$$u_{s,PWM} \in [1000, 2000] \begin{cases} 1200 \leq u_s < 1500 : & \text{turn left} \\ u_s = 1500 : & \text{drive straight} \\ 1500 < u_s \leq 1800 : & \text{turn right.} \end{cases}$$

We need to establish a model that maps the servo PWM signal to the resulting steering angle. However, we do not have a direct way of measuring the real steering angle; we can only measure the steady state yaw rate using the IMU sensor. Thus you will have to use a lateral kinematic model to convert your measured yaw rate and longitudinal speed to the corresponding steering angle when identifying your model.

1. *****Deliverable**: Derive the simple formula for calculating the steering angle as a function of the vehicle's longitudinal speed, wheelbase length, and yaw rate. You can use $L = 32cm$ as your wheelbase length.

2. Append the `lab4.launch` file to launch a node that automatically calls `SteeringMapping.py` when the launch file is fun. `SteeringMapping.py` sends a PWM step input of $u_s = 1400$ to the BARC's servo for five seconds before stopping the motor. This is enough time for the vehicle to converge to a steady-state yaw rate in response to your steering command.

3. Place your BARC vehicle on the ground, and ensure that there is approximately a 20 meter radius of free space around it. Designate a team member to move alongside the vehicle during the experiment, to prevent any collisions.

4. Begin the experiment by running the `lab4.launch` file:
   `roslaunch labs lab4.launch id:="your_experiment_name"`
   where you should replace `your_experiment_name` with your chosen experiment name.
   Make sure to specify different experiment names each time you call the launch file, so your previous trial's data do not get overwritten.
   When the vehicle has returned to standstill at the end of each experiment, end the data collection with `CTRL-C`.

5. Repeat this experiment for servo inputs in the range of $u_s = \{1400 : 20 : 1800\}$ (edit the `SteeringMapping.py` file as needed). You only need to do one experiment per servo input.

6. Import your experiment data into Matlab, and use the formula you derived in Step 1 to convert your collected longitudinal speed and yaw rate data into steering angle data.

7. *****Deliverable**: Submit a clean plot of your servo PWM input vs. resulting steering angle (in degrees). Fit a line of best-fit (in the time domain), and include the plotted equation in your figure.

8. Use the plot and your line of best-fit equation to determine what the true straight-ahead command is for your vehicle (i.e. what servo PWM command corresponds to a steering angle of 0?).

Task 4 Steering - Part 2 (No deliverables)

So far, you have only established a static relationship between servo PWM and the steering angle. The true lateral dynamics are composed of both steering actuator dynamics and vehicle dynamics, but we cannot separate these two dynamics from the inertial sensors without a steering angle sensor. This task explains how to create a makeshift steering angle sensor in order to estimate a first-order model of the steering dynamics, focusing in particular on the time delay between actuation command and steering response. **You do not need to complete this task, but we expect you understand what steps took place and utilize its results.**

1. The BARC camera is mounted above the vehicle, looking straight down at the front wheels. The camera is attached so it sits approximately one to two feet above the BARC, as in the setup depicted in Fig. 1.

2. We will use the camera to measure the steering angle of the vehicle in response to a reference command. In order to see the wheel response more clearly, we attach a dark pen to the front right wheel of the BARC using tape.

3. For this experiment, we will not send any motor commands to the BARC, focusing instead only on understanding the steering actuator dynamics. The vehicle is set on top of a box, so that the front wheels are suspended in the air and can turn freely in response to actuation commands.

4. The `lab4.launch` file is appended to launch the `SteeringDynamics.py` node. `SteeringDynamics.py` sends a PWM step input to steer straight for seven seconds before sending a new steering reference command for five seconds. For the first experiment, the new steering reference corresponds to $u_{s,PWM} = u_{0,PWM} + 25$, where $u_{0,PWM}$ is the vehicle's previously identified steer-straight command.

5. We need to record camera data during these experiments. In the `lab4.launch` file, the `camera_on` parameter value of the `record_experiment` node is changed to `"true"`.

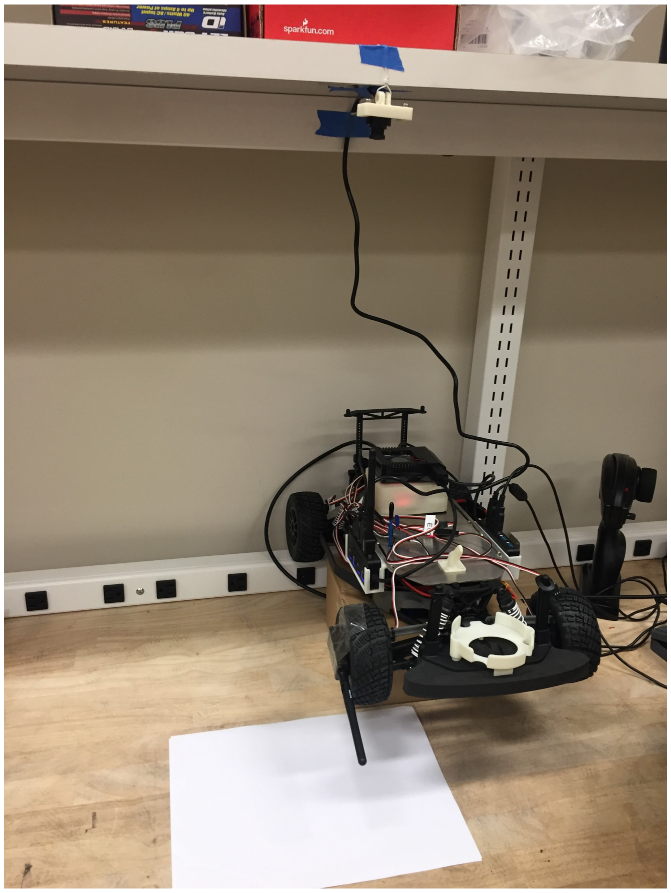


Figure 1: Experimental setup for the dynamic steering identification.

6. The edited `lab4.launch` file is launched with a provided experiment id. When the actuation inputs stop, the data collection is ended with `CTRL-C`.

7. This experiment is repeated for servo reference inputs of $u_{s,PWM} = u_{0,PWM} \pm \{25, 100, 150\}$, where $u_{0,PWM}$ is the vehicle's unique steer-straight command.

8. The experiment data is downloaded from the Dator server into Matlab. The video data will be a collection of frames taken at 30 frames per second, which can be manually label with a demonstrated steering angle as a function of time.

9. Fig. 2 plots the normalized servo PWM signal against the resultant steering angle. For simplicity, we will model the dynamic steering response as a pure time delay: $G_{pwm\rightarrow\delta}(s) = e^{-0.1s}$, where $\tau = 0.1[s]$ is the time-domain delay between the actuation command change and steering response.

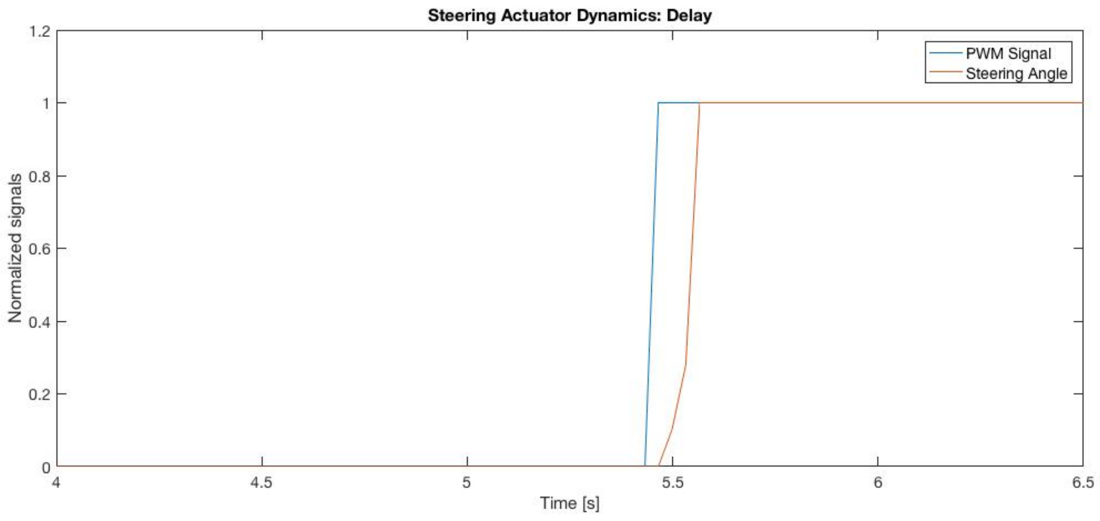


Figure 2

Task 5 Connecting to the BARC via wired connection (optional)

When there are wifi connectivity issues and you don't have an HMDI cable, it may be useful to connect directly to the car. You can use the following steps to set up the direct connection.

1. Use VNCViewer to connect to the odroid.
2. On the top right corner in Fig. 3, you should see the wicd-client icon. We are going to remove this network manager and install a different one.

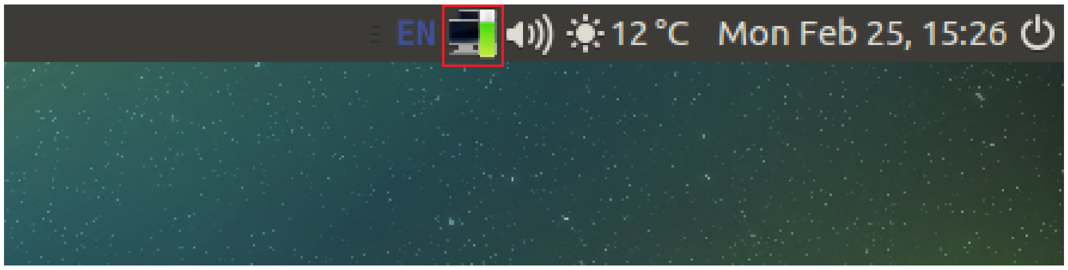


Figure 3: Wicd-Client

3. Make sure you are connected to a wireless network (CalVisitor or AirBears2 should work).
4. Pinging a website seems to help keep the wifi stable so run the following command in a terminal:
   `ping google.com`
   If it is successful, you should see a response such as "64 bytes from sfo03s01-in.net (216.50.100.200): icmp_seq=1 ttyl=53 time=21.0 ms". If the response is timing out, try reconnecting to another network. **Keep this running in the background!**
5. First, we want to install the new network manager before removing the old one. Open a new terminal and run the following command:
   `sudo apt-get install network-manager-gnome`

6. If that was successful, remove the wicd-client:
   `sudo apt-get purge wicd wicd-gtk`

7. Restart the odroid with
   `sudo reboot now`

8. Now on the top right corner in Fig. 4, a different icon should appear. Click it and select the **Edit Connections...** option.

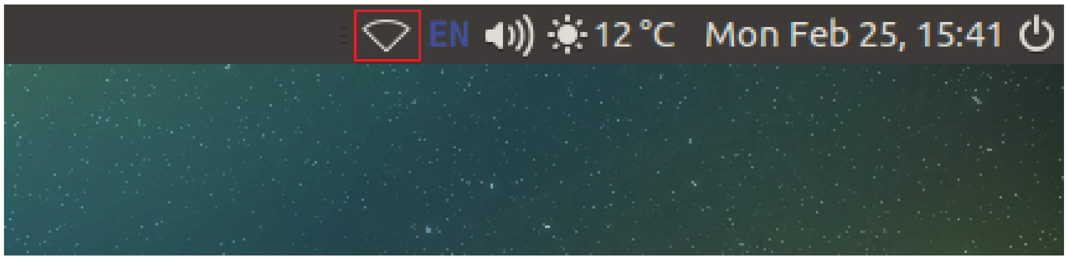
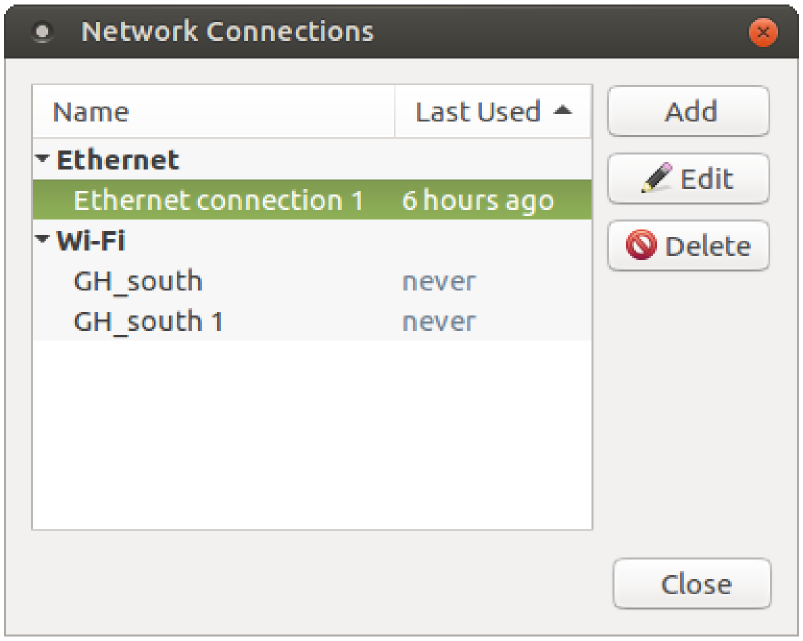


Figure 4: Gnome Network Manager

9. Select **Edit** on the first Ethernet and while **Ethernet** is the desired option, select **Create...**

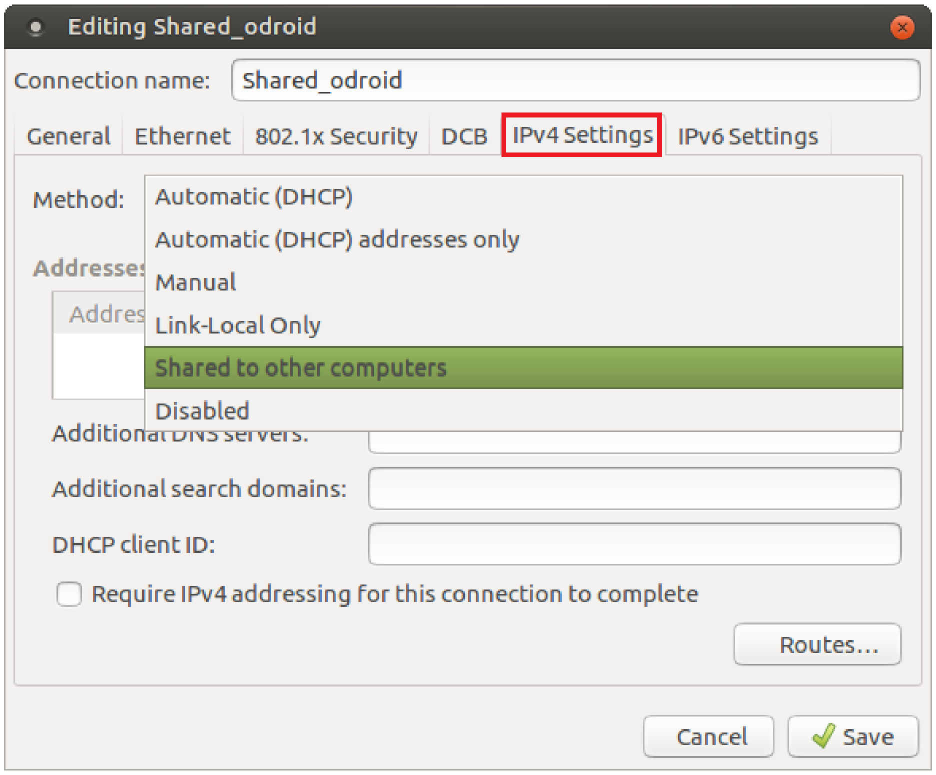


Name	Last Used
Ethernet	
Ethernet connection 1	6 hours ago
Wi-Fi	
GH_south	never
GH_south 1	never

Figure 5

10. Name the connection anything you'd like. Then, click **IPv4 Settings**.

11. Click the **Method** options and change it to **Shared to other computers**.



Editing Shared_odroid

Connection name: Shared_odroid

General | Ethernet | 802.1x Security | DCB | IPv4 Settings | IPv6 Settings

Method:
- Automatic (DHCP)
- Automatic (DHCP) addresses only
- Manual
- Link-Local Only
- Shared to other computers ✓
- Disabled

Additional search domains:

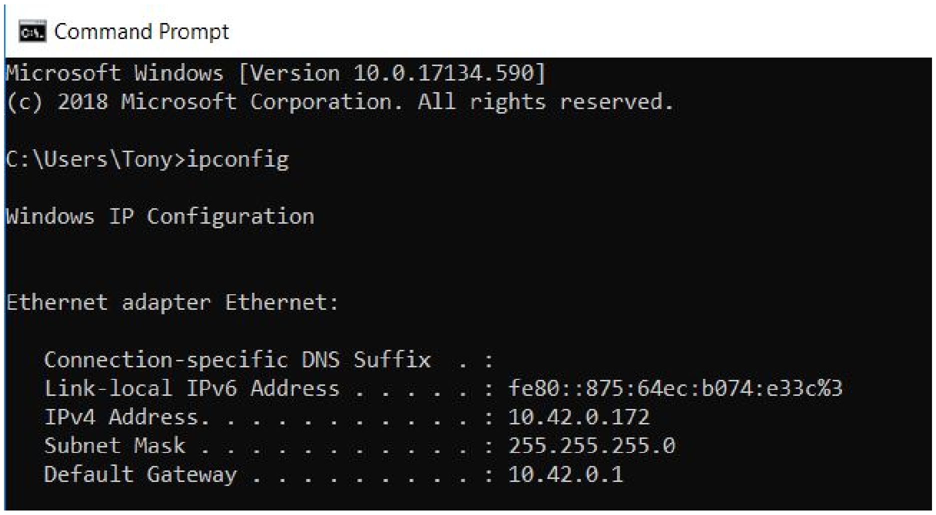
DHCP client ID:

☐ Require IPv4 addressing for this connection to complete

Figure 6

12. Save these settings and connect the odroid to your computer with the blue ethernet cable. On the top right corner of the odroid desktop, click your new created network.

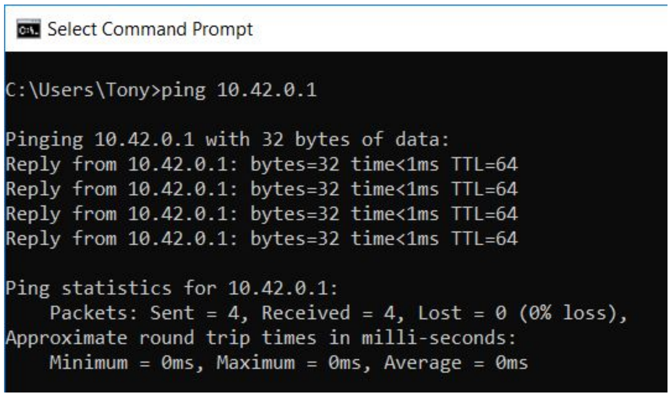
13. Now go back on the client computer (the one you are using VNCViewer from). On Windows, open the command prompt and run the command **ipconfig**. On MAC/Linux, run the command **ifconfig**. This will list the current network connections. You should see the "Default Gateway" address which will be the IP address of your new VNC connection.:



```
Command Prompt
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Tony>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::875:64ec:b074:e33c%3
   IPv4 Address. . . . . . . . . . . : 10.42.0.172
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.42.0.1
```

Figure 7

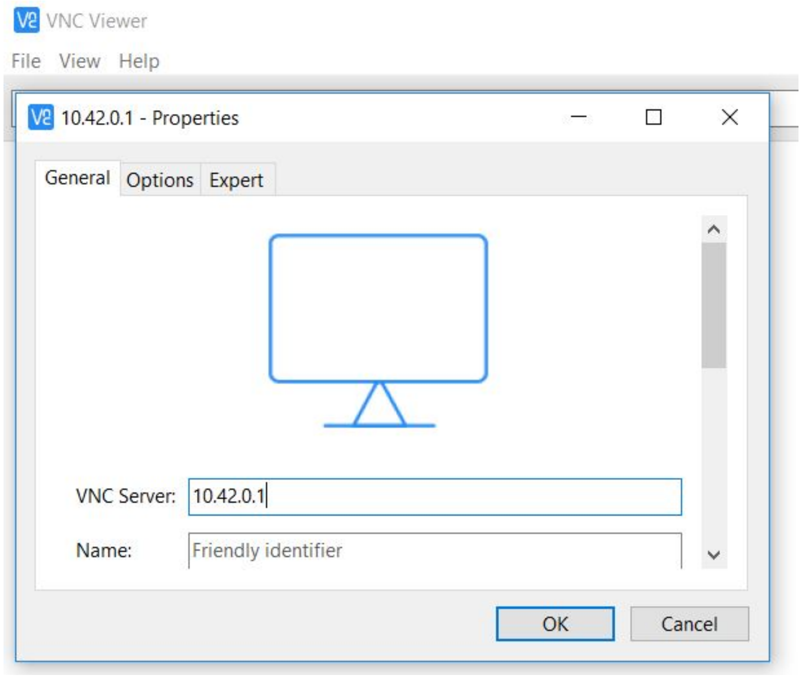
14. Ping the odroid to test the connection using:
    `ping 10.42.0.1`



```
Select Command Prompt

C:\Users\Tony>ping 10.42.0.1

Pinging 10.42.0.1 with 32 bytes of data:
Reply from 10.42.0.1: bytes=32 time<1ms TTL=64
Reply from 10.42.0.1: bytes=32 time<1ms TTL=64
Reply from 10.42.0.1: bytes=32 time<1ms TTL=64
Reply from 10.42.0.1: bytes=32 time<1ms TTL=64

Ping statistics for 10.42.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Figure 8

15. Run VNCViewer and create a new connection with **10.42.0.1** as the VNC Server IP address. You should be able to connect via wired or wireless connection now!



VNC Server: 10.42.0.1

Name: Friendly identifier

Figure 9