Task 0

- Create a launch file, that launches:
- 2 turtlesim_node from package turtlesim
- 1 turtle_teleop_key from package turtlesim
- You have to change the subscribed topic for one turtlesim, so teleop can drive only another turtlesim

  see http://wiki.ros.org/roslaunch/XML/remap for details

-
  ```
  git clone https://gitlab.com/osll/Duckietown-Software.git
  ```

-
  ```
  cd Duckietown-Software
  ```

-
  ```
  make
  ```

-
  ```
  cd catkin_ws
  ```

-
  ```
  catkin_make
  ```

-
  ```
  cd
  ```

-
  ```
  ssh-keygen -t rsa
  ```

-
  ```
  cat .ssh/id_rsa.pub | ssh ubuntu@duck.local 'cat >> .ssh/authorized_keys'
  ```
- In every terminal:

  ```
  export ROS_MASTER_URI="http://duck.local:11311"
  ```

  (instead of duck put duck2 or duck4)
- To launch bot use

  ```
  ./Duckietown-Software/utils/start_master_apriltags_any_intersection.sh duck
  ```

Task 1

- Create a publisher that allows robot to move forward.
- Robot is listening the topic duck/car_cmd_node/cmd
- to find the message_type use

  ```
  rostopic info duck/car_cmd_node/cmd
  ```

  use rosmsg show to find the fields of a message

The template for task 1

```
#!/usr/bin/env python
import rospy
from duckietown_msgs.msg import ***

if __name__ == '__main__':
  rospy.init_node('pub')
  pub = rospy.Publisher("duck/car_cmd_node/cmd", ***, queue_size=10)

  # fill msg

    pub.publish(msg)
```

Task 2

- Make a robot to follow some trajectory

- You can use the code from task_0 from the very first lesson (code in python without ROS) to keep the current position of the robot or you can harcode the trajectory and experimentally debug it.