Husky UGV Report

Ubuntu 20.04 Installation and Setting Up a Dual Boot Environment

Step-by-Step Guide

a) Download the Ubuntu 20.04 (Focal Fossa) Image

- Visit the official Ubuntu website to download the Ubuntu 20.04 ISO image: Ubuntu 20.04 Download.

b) Create a Bootable USB Stick

- To create the installation media, you need to write the downloaded ISO to a USB stick.
- Download and install the Rufus software from Rufus.
- Run Rufus, select the downloaded ISO file, and create the bootable USB stick.

c) Partition Your Storage Media

- Open the 'Create and format hard disk partitions' menu in the Windows Control Panel.
- Free up at least 50GB of space for the Ubuntu installation. Ensure you have enough space by shrinking an existing partition if necessary.

d) Disable Secure Boot and Encryption

- Before proceeding, disable Secure Boot and any disk encryption settings in your system's BIOS/UEFI settings. This step ensures compatibility during the installation process.

e) Insert the Bootable USB and Restart

- Insert the bootable USB stick into the laptop or PC where you want to install Ubuntu.
- Restart the device. It should automatically recognize the installation media. If it doesn't, hold the F12 key during startup to access the boot menu and select the USB device.

f) Start the Installation

- The Ubuntu Installation Menu will appear. Select your preferred language and choose the ‘Install Ubuntu’ option.
- Choose your keyboard layout or let Ubuntu detect it automatically.

g) Internet Connection and Installation Type

- Decide whether to connect to the Internet. For a faster installation, it's recommended not to connect.
- Select 'Normal Installation' if there is enough space available. Installing third-party software for graphics and Wi-Fi hardware, Flash, MP3, and other media is optional but can be beneficial.

h) Partitioning for Ubuntu Installation

- Choose the ‘Something Else’ option when the installation type window appears.
- Carefully select the free space you prepared earlier and set it as the root directory ("/").
- Click ‘Install Now’ and follow the prompts.

i) Time Zone and User Setup

- Select your time zone from the map or list.
- Set up your Ubuntu account by entering a username, password, and other relevant information.

j) Complete the Installation

- The installation process will begin and may take some time to complete.
- Once the installation is finished, you will have a dual boot environment where you can choose between Ubuntu and your existing operating system at startup.

Notes and Tips

- Ensure you back up important data before modifying partitions.
- If you encounter any issues with recognizing the USB stick, refer to your system's documentation for accessing the boot menu or BIOS/UEFI settings.
- After installation, you can customize Ubuntu and install additional software from the Ubuntu Software Center.

ROS Noetic Installation on Ubuntu 20.04 LTS

Resources:
https://hcr.cs.umass.edu/courses/compsci603/projects/Compsci_603_Project0_Preparation.pdf
http://wiki.ros.org/noetic/Installation/Ubuntu

Step-by-Step Guide

1. Update Ubuntu and Install Required Packages

Before installing ROS Noetic, ensure that your Ubuntu 20.04 LTS system is updated and has the necessary packages.

a) Update System Packages

- Open a terminal by pressing `Ctrl+Alt+T`.

Update your package lists and upgrade existing packages:
bash
Copy code
```
sudo apt update && sudo apt upgrade -y
sudo snap refresh
```

b) Install Development Packages

Install essential development tools:
bash
Copy code
```
sudo apt-get install build-essential gcc make cmake
```

2. Configure Ubuntu to Accept ROS Packages

To download ROS Noetic packages, you need to configure your system to accept software from `packages.ros.org`.

a) Add the ROS Package Repository

Add the ROS repository to your sources list:
bash
Copy code
```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
```

b) Set Up Your Keys

Install `curl` if it's not already installed:
bash
Copy code
```
sudo apt install curl
```

Add the ROS package keys:
bash
Copy code

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

3. Install ROS Noetic

a) Update Package Index

Ensure your package index is up-to-date:
bash
Copy code
```
sudo apt update
```

b) Choose ROS Installation Type

For a comprehensive installation, choose the Desktop-Full Install, which includes ROS, 2D/3D simulators, and perception packages:
bash
Copy code
```
sudo apt install ros-noetic-desktop-full
```

4. Environment Setup

a) Source ROS Setup Script

Source the ROS setup script in every bash terminal session:
bash
Copy code
```
source /opt/ros/noetic/setup.bash
```
To automatically source this script every time a new shell is launched, add it to your `.bashrc`:
bash
Copy code
```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

5. Install Additional ROS Tools and Dependencies

Install additional tools and dependencies needed for building ROS packages:
bash
Copy code
```
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
```

6. Initialize rosdep

`rosdep` is a command-line tool that helps you install system dependencies for building ROS packages.

a) Install rosdep

If you haven't installed `rosdep`, do so:
bash
Copy code
```
sudo apt install python3-rosdep
```

b) Initialize and Update rosdep

Initialize `rosdep`:
bash
Copy code
```
sudo rosdep init
rosdep update
```

Notes and Tips

- Always ensure your system is backed up before making significant changes like installing new packages.
- If you encounter any issues during installation, refer to the official ROS installation guide for troubleshooting tips and further instructions.

Tinker Twins Husky UGV Installation

Resources:
https://github.com/Tinker-Twins/Husky

Step-by-Step Guide

1. Installing Dependencies

Install the necessary ROS Noetic packages required for the Husky UGV:
Bash / open terminal
Copy following code

```
sudo apt-get install ros-noetic-fath-pivot-mount-description
sudo apt-get install ros-noetic-flir-camera-description
sudo apt-get install ros-noetic-velodyne-description
sudo apt install ros-noetic-realsense2-description
sudo apt-get install ros-noetic-lms1xx
sudo apt-get install ros-noetic-robot-localization
sudo apt-get install ros-noetic-interactive-marker-twist-server
sudo apt-get install ros-noetic-twist-mux
sudo apt-get install ros-noetic-teleop-twist-keyboard
sudo apt-get install ros-noetic-teleop-twist-joy
sudo apt-get install ros-noetic-rviz-imu-plugin
sudo apt-get install ros-noetic-gmapping

For joystick, use following

sudo apt-get install ros-noetic-joy
sudo apt-get install ros-noetic-teleop-twist-joy
sudo apt-get install ros-noetic-gazebo-ros
```

2. Setting Up the Catkin Workspace

Create a Catkin workspace if it doesn't already exist:
Bash / open terminal
Copy code

```
mkdir -p ~/catkin_ws/src
```

Navigate to the source directory of your Catkin workspace:
bash
Copy code

```
cd ~/catkin_ws/src
```

3. Cloning the Husky Repository

Clone the Tinker Twins Husky repository:
bash
Copy code

```
git clone https://github.com/Tinker-Twins/Husky.git
```

4. Building the Packages

Navigate back to the Catkin workspace root:
bash
Copy code
```
cd ~/catkin_ws
```

Build the packages using `catkin_make`:
bash
Copy code
```
catkin_make
```

5. Usage

a) Keyboard Teleoperation

Launch the Husky in Gazebo:
bash
Copy code
source devel/setup.bash
```
roslaunch husky_gazebo husky_playpen.launch
```

Launch the teleoperation keyboard control:
bash
Copy code
source devel/setup.bash
```
roslaunch husky_control teleop_keyboard.launch
```

`Xbox Controller`

```
rosrun joy joy_node
rosrun teleop_twist_joy teleop_node
```

```
How to use Xbox to control the movement of the husky ??
While holding A, Move the left or right joystick according to your configuration.
```

b) Map-Less Navigation

Launch the Husky in Gazebo:
bash
Copy code
```
roslaunch husky_gazebo husky_playpen.launch
```

Launch the robot visualization:
bash
Copy code
```
roslaunch husky_viz view_robot.launch
```

Launch map-less navigation:
bash
Copy code
```
roslaunch husky_navigation map_less_navigation.launch
```

c) Simultaneous Localization and Mapping (SLAM)

Launch the Husky in Gazebo:
bash
Copy code
```
roslaunch husky_gazebo husky_playpen.launch
```

Launch the robot visualization:
bash
Copy code
```
roslaunch husky_viz view_robot.launch
```

Launch the gmapping for SLAM:
bash
Copy code
```
roslaunch husky_navigation gmapping.launch
```

Launch the teleoperation keyboard control:
bash
Copy code
```
roslaunch husky_control teleop_keyboard.launch
```

To save the generated map to the current working directory:
bash
Copy code
```
rosrun map_server map_saver -f <filename>
```

d) Adaptive Monte Carlo Localization (AMCL)

Launch the Husky in Gazebo:
bash
Copy code
```
roslaunch husky_gazebo husky_playpen.launch
```

Launch the robot visualization:
bash
Copy code
```
roslaunch husky_viz view_robot.launch
```

Launch AMCL for localization:
bash
Copy code
```
roslaunch husky_navigation amcl.launch
```

Launch the teleoperation keyboard control:
bash

Copy code

```
roslaunch husky_control teleop_keyboard.launch
```

e) Map-Based Navigation

Launch the Husky in Gazebo:
bash
Copy code

```
roslaunch husky_gazebo husky_playpen.launch
```

Launch the robot visualization:
bash
Copy code

```
roslaunch husky_viz view_robot.launch
```

Launch map-based navigation:
bash
Copy code

```
roslaunch husky_navigation map_based_navigation.launch
```

Notes and Tips

- Ensure you have ROS Noetic installed and properly configured before starting with the Husky UGV installation.
- Familiarize yourself with basic ROS commands and concepts to effectively use and troubleshoot the system.
- Refer to the Tinker Twins Husky GitHub repository for more detailed documentation and updates.

Launching Husky in a Custom Gazebo World

This section explains how to create a custom world in Gazebo and launch the Husky robot within it. This section should be added after the **Simulate a Trained DMP Trajectory in Gazebo** section.

Steps to Build and Launch a Custom World in Gazebo

Build a Custom World in Gazebo:

Open Gazebo:
bash
Copy code
`gazebo`

1. **Adjust the View**:
    - Use the view controls to adjust the view angle and position to your preference.
2. **Enter Building Editor Mode**:
    - Click on **Edit**.
    - Select **Building Editor**.
3. **Create the Building Structure**:
    - Use the tools to create walls. You can add features like windows, doors, and stairs.
    - Add textures to the walls to enhance realism.
4. **Save the Model**:
    - Click on **File**.
    - Select **Save As**.
    - Save the model in the `catkin_ws/src/Husky/husky_gazebo/models` directory.
    - Enter a name for the model and click **Save**.
5. **Exit the Model Builder**:
    - Click on **Edit**.
    - Select **Exit Model Builder**.
6. **Add Other Elements**:
    - Use the **Insert** tab to add other elements like furniture, vehicles, or other objects.
7. **Save the World**:
    - Click on **File**.
    - Select **Save World As**.
    - Save the world in the `catkin_ws/src/Husky/husky_gazebo/worlds/` directory.
    - Enter a name for the world and click **Save**.
8. Congratulations, you have successfully created a custom world in Gazebo!

Create a Launch File for Your Custom World:

Navigate to the Launch Directory:
bash
Copy code
`cd ~/catkin_workspace/src/Husky/husky_gazebo/launch`

9. **Create a New Launch File**:
    - Create a new file, e.g., `custom_world.launch`:

bash
Copy code
`nano custom_world.launch`

Add the Following Code to the Launch File:
xml
Copy code

```
<launch>
    <arg name="world_name" default="$(find husky_gazebo)/worlds/custom_world.world"/>
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(arg world_name)"/>
        <arg name="paused" value="false"/>
        <arg name="use_sim_time" value="true"/>
        <arg name="gui" value="true"/>
        <arg name="headless" value="false"/>
        <arg name="debug" value="false"/>
    </include>
    <include file="$(find husky_gazebo)/launch/spawn_husky.launch"/>
</launch>
```

10. **Save and Close the File**:
    - Save the file and exit the editor.

Rebuild the workspace using catkin_make:
Open a Terminal:
bash
Copy code

```
cd ~/catkin_workspace

catkin_make
```

Launch the Custom World with Husky:
Open a Terminal:
bash
Copy code

```
cd ~/catkin_workspace

source devel/setup.bash
```

Run the Launch File:
bash
Copy code

```
roslaunch husky_gazebo custom_world.launch
```

11. This will launch Gazebo with your custom world and spawn the Husky robot in it. You can control the Husky as described in the previous sections, using either the keyboard or joystick and record its trajectory.

Controlling and Recording Trajectories for Husky Mobile Robot in Gazebo

This section describes how to record the trajectory of a Husky mobile robot in Gazebo, including position and velocity data, and then train and execute a DMP model to follow the recorded trajectory.

Record the Trajectory

Create a New Package for Recording Trajectory Data:

bash
Copy code

```
cd ~/catkin_workspace/src
```

```
catkin_create_pkg trajectory_recorder rospy std_msgs
```

1. **Create a Python Script (trajectory_recorder.py) to Record Trajectory**:
   - This script will create a node that subscribes to the `/odometry` topic to record the position and velocity of the Husky.
   - Save the script to `src/trajectory_recorder`.
   - Create a `record.csv` file in the same folder.
   - Script link: trajectory_recorder.py

2. **Make the Node Executable**:
   bash
   Copy code
   ```
   chmod +x src/trajectory_recorder/trajectory_recorder.py
   ```

3. **Update CMakeLists.txt and package.xml**:

CMakeLists.txt:
cmake
Copy code

```
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  nav_msgs
)
```

package.xml:
xml
Copy code

```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>nav_msgs</build_depend>
<build_depend>rospy</build_depend>
```

```
<build_depend>std_msgs</build_depend>
<build_export_depend>nav_msgs</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>nav_msgs</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

4. **Build the Workspace**:
   bash
   Copy code
   ```
   cd ~/catkin_workspace
   catkin_make
   ```

5. **Source the Workspace Again**:
   bash
   Copy code
   ```
   source devel/setup.bash
   ```

6. **Launch the Husky in Gazebo**:
   bash
   Copy code
   ```
   roslaunch husky_gazebo husky_playpen.launch
   ```

7. **Control the Husky Using an Xbox Controller**:

   Open a new terminal and run:
   bash
   Copy code
   ```
   rosrun joy joy_node
   ```
   Open another terminal and run:
   bash
   Copy code
   ```
   rosrun teleop_twist_joy teleop_node
   ```

   - To control the movement, hold the **A** button and move the left or right joystick according to your configuration.

8. **Run Your Node to Record the Trajectory**:
   bash

Copy code

```
source devel/setup.bash
```

```
rosrun trajectory_recorder trajectory_recorder.py
```

```
or
```

```
rosrun trajectory_recorder Angular_recorded.py
```

Once you are done recording, press `Ctrl+C` to stop the node. The recorded data will be saved in the `record.csv` file specified in the Python script.

9. Run the node for replaying the csv file recorded in the previous step :

Link for the replay script : replay_trajectory.py

Note : Relaunch gazebo with the same environment used while recording in new terminal

bash/terminal

Copy code

```
source devel/setup.bash
roslaunch husky_gazebo husky_playpen.launch
~/catkin_ws/src/trajectory_recorder$ chmod +x replay_trajectory.py
rosrun trajectory_recorder replay_trajectory.py
```

After, Running this node you will see your husky robot moving and following the same trajectory that was feeded in the csv file.

Train and Execute the DMP Model*

1. **Train a DMP Model**:
    - **Use Python to Train a DMP Model**:

Install necessary libraries:

bash

Copy code

```
pip install numpy pandas
```

Optional: you can use an inbuilt DMP package:

bash

Copy code

```
pip install numpy pandas pydmps
```

- **Create a Script to Train the DMP Model**:
    - Using the recorded CSV file, link to the DMP algorithm and genetic algorithm used to find the best hyperparameters: DMP Trajectory Training
    - Use VSCode to install the Python Extension Pack.

Check the script for required imports:
python
Copy code

```
import sys
import time
import rospy
from math import pi
from std_srvs.srv import Empty
import numpy as np
import matplotlib.pyplot as plt
from sensor_msgs.msg import JointState
import os
import pandas as pd
from geometry_msgs.msg import PoseStamped
import itertools
import random
import math
import csv
from termcolor import colored
```

Run the Python Script:
bash
Copy code

```
cd ~/catkin_workspace/src/trajectory_recorder
python3 DMP_trajectory_training.py
```

- **Note**: After applying this algorithm, the DMP will be trained to follow the trajectory, producing the best hyperparameters for simulation in Gazebo.
  - Example parameters: `(ax, ax_new, beta, alpha): (10.756927952392, 4.385437285094811, 13.409911624757482, 19.317307692733, 101)`

2. **Simulate a Trained DMP Trajectory in Gazebo**:
   - **Generate the Trajectory Using DMP Parameters**:
     - Create a script that generates a trajectory using the best parameters produced above.
     - Link: Generate Trajectory Script
   - **Publish the Trajectory as ROS Messages**:
     - Create a ROS node to publish the trajectory.
     - Link: Publish Trajectory Node

Make the script executable:
bash
Copy code
```
chmod +x publish_trajectory.py
```

Source the Workspace Again:
bash
Copy code
```
source devel/setup.bash
```

Launch the Husky in Gazebo:
bash
Copy code
```
roslaunch husky_gazebo husky_playpen.launch
```

Run the Node for Publishing ROS Control Messages to Simulate in Gazebo:
bash
Copy code
```
rosrun trajectory_recorder publish_trajectory.py
```

3. **Execute the Trajectory in Gazebo**:
    - **Create a ROS Node That Subscribes to the /husky/dmp_trajectory Topic**:
        - This node sends appropriate movement commands to the robot in Gazebo.
        - Link: Execute Trained DMP Trajectory Node

Run the Node:
bash
Copy code
```
source devel/setup.bash
```

```
chmod +x TrainedDMP_trajectory_execute.py
```

```
rosrun trajectory_recorder TrainedDMP_trajectory_execute.py
```

- **Note**: After completing these steps, you will see your Husky robot moving in the Gazebo environment. You can adjust the speed through changes in the execute node created in the above step.

Physical Husky Robot

Firstly, To start working with the husky, We need to connect it through any hotspot via ip address of your mobile phone or any system.

Steps to connect it through any hotspot :

`cd /etc/netplan` # Change the directory

`nano 00-installer-config.yaml` # Add your network name and it's hotspot address to connect the husky and save.

`sudo netplan apply` # To save the changes done in the above file through nano editor

To see the ip address of the connected hotspot :

`ip a`

To control the husky movement through the connected joystick :

`roslaunch husky_control teleop_joystick.launch`

Now, To control the husky robot with your pc/laptop, We have to connect it's wifi with the same network that is connected with husky and thereafter to load it's administrator terminal in our system, We have to connect through ssh via ip address……..

Open system terminal to connect with physical husky :

`ssh administrator@192.168.78.133` #You have to enter the ipv4 address of your connected hotspot, You can check it through `ip a`

Then enter password : robotics

Now, You will get some message where it will be mentioned that last login XX/XX/XXXX, It means you are connected with the husky for any operation through your system wirelessly.

Record the Trajectory

Create a New Package for Recording Trajectory Data:

bash

Copy code

```
cd ~/husky_ws/src
```

```
catkin_create_pkg trajectory_recorder rospy std_msgs
```

1. **Update CMakeLists.txt and package.xml in that custom package created above** :

CMakeLists.txt:
cmake
Copy code

```
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  nav_msgs
)
```

package.xml:
xml
Copy code

```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>nav_msgs</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>nav_msgs</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>nav_msgs</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

2. Create a Python Script (trajectory_recorder.py) to Record Trajectory:

- This script will create a node that subscribes to the `/odometry` topic to record the position and velocity of the Husky.
- Save the script to `src/trajectory_recorder`.
- Create a `record.csv` file in the same folder.
- Script link: trajectory_recorder.py

3. Make the Node Executable:
bash
Copy code

```
chmod +x src/trajectory_recorder/trajectory_recorder.py
```

4. Create record.csv file for recording the data into it :

```
cd ~/husky_ws/src/trajectory_recorder
nano record.csv
```

Then just save and exit the csv file.

5. Run Your Node to Record the Trajectory :

```
cd ~/husky_ws
```

```
source devel/setup.bash
rosrun trajectory_recorder trajectory_recorder.py
```

Now, Use your joystick/keyboard to move husky along any trajectory of your choice and once done with that press **ctrl+c** to terminate the process

6. Run the node for replaying the csv file recorded in the previous step :

Link for the replay script : replay_trajectory.py

Now, Create a python script in your package wih any suitable name for replaying the csv file and positioned your husky robot to the position at the starting of recording task previously.

bash/terminal

```
source devel/setup.bash
~/husky_ws/src/trajectory_recorder$ chmod +x replay_trajectory.py
rosrun trajectory_recorder replay_trajectory.py
```

After running the above node, You will see your husky robot moving along the same trajectory autonomously

TO launch husky with kinova robotic arm

https://github.com/husky/husky_manipulation