Section 2

CS237A Fall 2025
Last updated Oct 5, 2025

Overview

The goals for this section:

1. Create a ROS Workspace.
2. Create a ROS Package.
3. Create a ROS Node with a publisher and a subscriber.

Prep

Task 0.1 — Cleanup

Close any open windows and terminals. Remove directory `~/autonomy_ws` if it exists. This might be leftover from a previous section.

Task 0.2 — GitHub Account

Use the following commands to logout of any existing GitHub accounts, and login to the account of one of your group members.

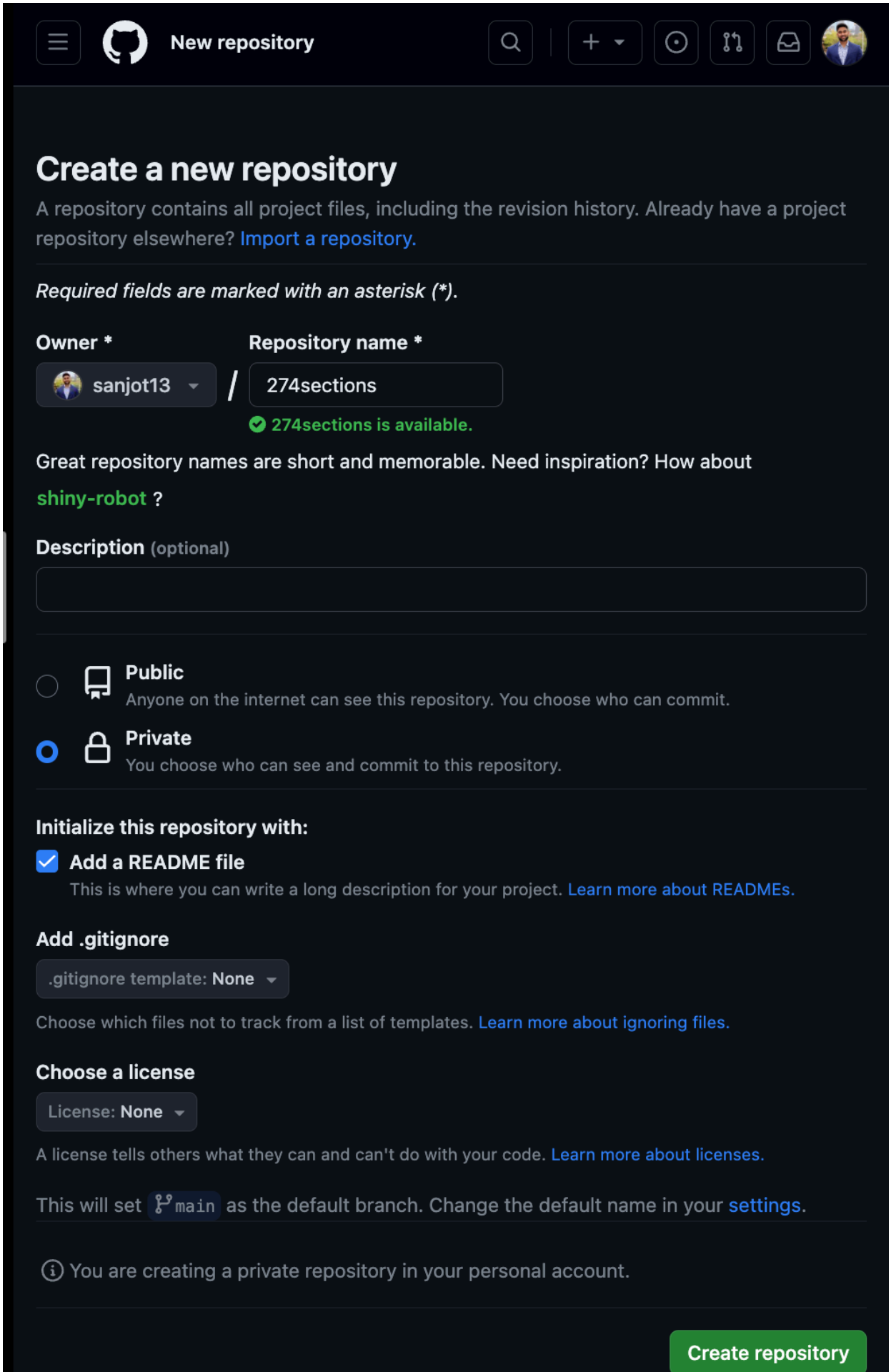
```shell
gh auth logout  # Log out of any accounts from previous sections.

gh auth login   # Login to the account your group is using.
# (1) press enter to select Github.com
# (2) select press enter to select HTTPS
# (3) type Y then hit enter
# (4) Take note of the supplied code, then press enter to open
```

```
#     the web browser. Follow the instructions in the browser to
#     login to your GH account.
# (5) Once you've logged in, return to the terminal and press enter
#     to finalize the authentication process.
```

Task 0.3 — Group Repo

If you haven't already, go to github.com and create a *private* repository for your group (make sure to initialize this repository with a `README` file), and add the other members of your group as collaborators (to do this go to Settings > Collaborators in your repo once you've created it).


New repository
Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.
Required fields are marked with an asterisk (*).
Owner *
Repository name *
sanjot13
/
274sections
274sections is available.
Great repository names are short and memorable. Need inspiration? How about shiny-robot ?
Description (optional)
Public
Anyone on the internet can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with:
Add a README file
This is where you can write a long description for your project. Learn more about READMEs.
Add .gitignore
.gitignore template: None
Choose which files not to track from a list of templates. Learn more about ignoring files.
Choose a license
License: None
A license tells others what they can and can't do with your code. Learn more about licenses.
This will set main as the default branch. Change the default name in your settings.
You are creating a private repository in your personal account.
Create repository

Create a ROS Workspace

ROS Workspaces are used to organize packages that are under development. Here your group will create one that will be used during this section. Complete the following tasks using terminal commands **only** on the provided Skilling Laptops.

Task 1.1 — Directories
Create the base directory structure for your group's ROS workspace, `~/autonomy_ws/src`.

Task 1.2 — Clone and Branch
Clone your group's GitHub repo to the `src/` directory, and create a new branch called `section2` .

Task 1.3 — Create a Package
Create an empty ROS package inside of your group's git repository with the name `s2_basic`.

Task 1.4 — Build
Navigate back to the "root" of your workspace, `~/autonomy_ws/` , and build your ROS workspace using the command `colcon build` (do **not** run this command in any other directory). You should see something like the output below.

```
Shell
Starting >>> s2_basic
Finished <<< s2_basic [0.87s]

Summary: 1 packages finished [0.87s]
```

Cheat Sheet:

```shell
ls                      # List files and folders in current directory
cd <directory>          # Change into directory
mkdir -p <directory>    # Make a directory with a given name
git clone <url>         # Clone a Git repository from a URL
git checkout -b <name>  # Create a branch in a repository
git checkout <name>     # Switch to a branch with a given name

# Create a new ROS package with the CMake build system.
ros2 pkg create --build-type ament_cmake <package_name>

touch <file>     # create a file
chmod +x <file>  # make the file an executable

code <file>   # edit a file with VSCode
vim <file>    # edit a file with vim
```

Folder Structure:

```shell
# Directory organization for a ROS Development Workspace

~/tb_ws/     # My group's workspace root directory
        build/             # Compiled libraries
        install/setup.bash # Source workspace packages
        src/               # Top-level package directory
                    external_pkg1/  # Some external package that I am using
                    external_pkg2/   # Another external package
                    group4_repo/     # My group's autonomy repository
                           navigation_pkg/    # My group's Navigation ROS
Package
                           perception_pkg/    # My group's Perception ROS
Package
                                  CMakeLists.txt  # Build instructions
                                  package.xml     # ROS package details
                                  src/            # C++ Code lives here!
                                  my_pytorch_py_lib/   # My custom python module
                                  scripts/             # ROS Python Executables
                                         camera_node.py     # Camera node code
                                         detection_node.py  # Detection node
code
```

Create a ROS Node

Task 2.1 — Create The File
Create a file (and directory if necessary) at `s2_basic/scripts/constant_control.py`. Don't forget to make this an executable (Hint: use `chmod`)!

Task 2.2 — Write a ROS Node
Edit the file to create a ROS node that periodically prints the message "sending constant control..." every 0.2 seconds. Refer to the cheat sheet section if you don't know where to start.

Hint: you will need a timer, take a look at `create_timer` in the example below.

Task 2.3 — Register Your Node with CMake
Add the following lines to `s2_basic/CMakeLists.txt`.

```
Shell
install(PROGRAMS
    scripts/constant_control.py
    DESTINATION lib/${PROJECT_NAME}
)
```

Task 2.4 — Edit your package.xml
Add the following lines which add runtime dependencies for `rclpy` and `std_msgs`.

```
Shell
colcon build --symlink-install
```

Important Note: *Your current working directory in your terminal must be on `~/autonomy_ws` when building, otherwise weird directories could pop up inside your repository.*

Task 2.6 — Run Your ROS Node
First source your autonomy workspace. This is required whenever you

open a new terminal and want to run scripts from your autonomy workspace.

```shell
source ~/autonomy_ws/install/setup.bash
```

Then run the following command and replace `<...>` with the appropriate string to execute your ROS node.

```shell
ros2 run <package_name> <script_file>
```

Hint: Both package name and script file is mentioned somewhere earlier.

In order to see if your publisher is working correctly, open a new terminal and run the following command to view the topic you are publishing to:

```shell
ros2 topic echo <topic_name>
```

Checkpoint — Call a CA over to demonstrate your periodic printing node!

Cheat Sheet

You can check this for an example of a minimal ROS2 node. You can also use the simple heartbeat node below as a guide for implementing later parts of the section.

```shell
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
```

```
# import the message type to use
from std_msgs.msg import Int64, Bool


class Heartbeat(Node):
    def __init__(self) -> None:
       # initialize base class (must happen before everything else)
       super().__init__("heartbeat")

       # a heartbeat counter
       self.hb_counter = 0

       # create publisher with: self.create_publisher(<msg type>, <topic>,
<qos>)
       self.hb_pub = self.create_publisher(Int64, "/heartbeat", 10)

       # create a timer with: self.create_timer(<second>, <callback>)
       self.hb_timer = self.create_timer(1.0, self.hb_callback)

       # create subscription with: self.create_subscription(<msg type>, <topic>,
<callback>, <qos>)
       self.motor_sub = self.create_subscription(Bool, "/health/motor",
self.health_callback, 10)

    def hb_callback(self) -> None:
       """
       Heartbeat callback triggered by the timer
       """
       # construct heartbeat message
       msg = Int64()
       msg.data = self.hb_counter

       # publish heartbeat counter
       self.hb_pub.publish(msg)

       # increment counter
       self.hb_counter += 1

    def health_callback(self, msg: Bool) -> None:
        """
        Sensor health callback triggered by subscription
        """
        if not msg.data:
            self.get_logger().fatal("Heartbeat stopped")
            self.hb_timer.cancel()


if __name__ == "__main__":
    rclpy.init()        # initialize ROS2 context (must run before any other
rclpy call)
```

```
    node = Heartbeat()  # instantiate the heartbeat node
    rclpy.spin(node)    # Use ROS2 built-in schedular for executing the node
    rclpy.shutdown()    # cleanly shutdown ROS2 context
```

Send Out Constant Control with Publisher

Task 3.1 — Modify Your Node

Work on your created ROS node and make it publish a constant `Twist`(click link for more info) message to the `/cmd_vel` topic periodically. The `Twist` message has the following data structure:

```
Shell
Twist
  angular
    x
    y
    z
  linear
    x
    y
    z
```

Hint:

1. Keep your timer, but replace the `print` statement with your publishing logic.
2. Refer to the cheat sheet above for some examples on how to
    1. import message types from library
    2. create and initialize a publisher
    3. construct messages
    4. publish messages
3. `Twist` message type comes from `geometry_msgs.msg`.
4. TurtleBot is a 2D robot with differential drive dynamics, and therefore, you only control a linear velocity and an angular velocity, i.e. you only need to set two fields in the `Twist` message:

```
Shell
msg = Twist()  # 0 initialize everything by default
msg.linear.x = ...  # set this to be the linear velocity
msg.angular.z = ... # set this to be the angular velocity
```

(Optional Checkpoint) — If you are unsure about your implementation, call up a CA to help check on it.

Task 3.2 — Run in Simulation

First launch the simulator in a separate terminal by running the following command.

```
Shell
ros2 launch asl_tb3_sim signs.launch.py
```

Inspect the ROS2 topics that are used by the simulator with the following command, and you should see a list of topics including `/cmd_vel`.

```
Shell
ros2 topic list
```

In a separate terminal,

1. Source the install script for autonomy workspace (Hint: recall from previous step).
2. Run your ROS node (Hint: recall from previous step).

Checkpoint — Call up a CA and demonstrate that your robot is moving in sim!

Emergency Break

Task 4.1 — Create a Subscription

Work on the same node and create a subscription on the topic `/kill`

with a message type of `std_msgs.msg.Bool`. Create the subscription callback so that whenever you receive a `True` message from `/kill`,

1. Stop the timer (which effectively stops the periodic publication to the topic `/cmd_vel`).
2. Publish a 0 control to `/cmd_vel`.

Hint: refer to the cheat sheet for examples on how to

1. create and initialize a subscription.
2. create a callback function.

(Optional Checkpoint) — If you are unsure about your implementation, call up a CA to help check on it.

Task 4.2 — Test in Simulation
Recall from Task 3.2, launch the simulation first and then run your constant control node. After it is running for a while, publish a kill message with the following command

```shell
ros2 topic pub /kill std_msgs/msg/Bool data:\ true -1
```

Checkpoint — Call up a CA and demonstrate that your robot stops on a kill message!

Wrap Up

1. Push your code(add, commit, and push) to your section2 branch!

```shell
git add s2_basic/             # add the 's2_basic/' folder
git commit -m "<message>"     # commit your changes to the local branch
git push -u origin section2   # push for the first time after creating a branch
git push                      # subsequent pushes on the same branch
```

2. Create a PR either from Github web page or by running `gh pr create` in your terminal.

Checkpoint — Call a CA over to show your PR and sign you out!