Web_Nav

February 13, 2023

[ ]:
```
To control a robot using a web page, you can create a web application that␣
  ↪communicates with the robot through the ROS network. Here are the general␣
  ↪steps to do this:

Set up a web server: You can set up a web server on a machine that is connected␣
  ↪to the same network as the robot. You can use a web framework like Flask or␣
  ↪Django to serve the web pages and handle HTTP requests.

Create a web interface: You can use HTML, CSS, and JavaScript to create a user␣
  ↪interface for controlling the robot. You can add buttons, sliders, or other␣
  ↪controls to the interface, and use JavaScript to send commands to the robot␣
  ↪when these controls are activated.

Connect to the ROS network: You can use a web socket library like rosbridge to␣
  ↪connect to the ROS network and send commands to the robot. Rosbridge␣
  ↪provides a JSON interface to ROS, allowing you to send ROS commands and␣
  ↪receive feedback from the robot using simple HTTP requests.

Send commands to the robot: You can use the JavaScript code in your web␣
  ↪interface to send commands to the robot over the ROS network. For example,␣
  ↪you could send a command to move the robot's arm or turn on a camera when a␣
  ↪button is clicked.

Receive feedback from the robot: You can also receive feedback from the robot␣
  ↪and update the web interface in real-time. For example, you could display␣
  ↪the robot's current position or the video feed from a camera.

By combining these steps, you can create a web-based interface for controlling␣
  ↪a robot through the ROS network. This allows you to control the robot from␣
  ↪anywhere with an internet connection and a web browser.

Here are the steps to implement a web GUI with buttons that control a robot in␣
  ↪ROS:

Set up a web server: You can set up a web server using a web framework like␣
  ↪Flask or Django. You can serve HTML pages and handle HTTP requests with this␣
  ↪framework.
```

```
Create a web interface: Use HTML, CSS, and JavaScript to create a web interface␣
 ↪with buttons. You can create a form with buttons or use other UI elements␣
 ↪like buttons or links to trigger actions.

Connect to ROS: You can use a web socket library like rosbridge to connect to␣
 ↪the ROS network and send messages to the robot. You can set up a rosbridge␣
 ↪server on the same machine as the web server, and connect to it from your␣
 ↪web interface using JavaScript.

Define a ROS topic: In ROS, you can define a topic that the robot can subscribe␣
 ↪to in order to receive messages. For example, you can define a topic called␣
 ↪"robot_pose" that the robot can subscribe to in order to receive the target␣
 ↪pose it should go to.

Publish messages to the topic: When a button is clicked in the web interface,␣
 ↪you can use JavaScript to send a message to the ROS network with the target␣
 ↪pose for the robot. You can do this by publishing a message to the␣
 ↪"robot_pose" topic that was defined in step 4.

Receive messages in ROS: On the robot, you can write a ROS node that subscribes␣
 ↪to the "robot_pose" topic. When it receives a message, it can extract the␣
 ↪target pose from the message and use it to navigate to the target location.

Store map data: You can store the map data in a ROS bag file, which is a data␣
 ↪format for storing and playing back ROS messages. You can use the ROS bag␣
 ↪file to save the map data and then load it onto the robot when it starts up.

By following these steps, you can create a web interface with buttons that␣
 ↪control a robot in ROS. You can publish messages to a ROS topic with the␣
 ↪target pose for the robot, and the robot can use this information to␣
 ↪navigate to the target location.

Here is an example of a ROS node written in Python that subscribes to the␣
 ↪"robot_pose" topic and uses the target pose to navigate to the target␣
 ↪location:
```

[ ]:
```python
# Import the necessary libraries
import rospy
from geometry_msgs.msg import Pose

def pose_callback(pose_message):
    # Extract the target pose from the message
    target_pose = pose_message.pose

    # Use the target pose to navigate to the target location
    navigate_to_target(target_pose)

def navigate_to_target(target_pose):
    # Code to navigate the robot to the target location using the target pose
    # ...

if __name__ == '__main__':
    rospy.init_node('navigation_node')

    # Subscribe to the "robot_pose" topic
    rospy.Subscriber("robot_pose", Pose, pose_callback)

    # Spin to keep the node running
    rospy.spin()
```

[ ]:
```
In this example, the pose_callback function is called every time a message is␣
  ↪received on the "robot_pose" topic. The function extracts the target pose␣
  ↪from the message and passes it to the navigate_to_target function. The␣
  ↪navigate_to_target function implements the logic to navigate the robot to␣
  ↪the target location using the target pose.

You can modify the navigate_to_target function to include your own code to␣
  ↪control the robot and navigate it to the target location. For example, you␣
  ↪could use a navigation stack like the ROS Navigation Stack to plan a path␣
  ↪for the robot and execute the path.
----
To pass the target pose to the ROS Navigation Stack, you will need to publish␣
  ↪it to a topic that the navigation stack is subscribed to. In the ROS␣
  ↪Navigation Stack, the common topic used for publishing goals is "/
  ↪move_base_simple/goal".

Here's an updated version of the code to publish the target pose to the "/
  ↪move_base_simple/goal" topic:
```

[ ]:
```
# Import the necessary libraries
import rospy
from geometry_msgs.msg import PoseStamped

def pose_callback(pose_message):
    # Extract the target pose from the message
    target_pose = pose_message.pose

    # Convert the target pose to a PoseStamped message
    goal = PoseStamped()
    goal.header.stamp = rospy.Time.now()
    goal.header.frame_id = "map"
    goal.pose = target_pose
    # Publish the goal to the "/move_base_simple/goal" topic
    goal_publisher.publish(goal)

if __name__ == '__main__':
    rospy.init_node('navigation_node')

    # Subscribe to the "robot_pose" topic
    rospy.Subscriber("robot_pose", Pose, pose_callback)

    # Create a publisher for the "/move_base_simple/goal" topic
    goal_publisher = rospy.Publisher("/move_base_simple/goal", PoseStamped,␣
↪queue_size=10)

    # Spin to keep the node running
    rospy.spin()
```

```
[ ]: In this updated code, the target pose is converted to a PoseStamped message and␣
     ↪published to the "/move_base_simple/goal" topic using the goal_publisher␣
     ↪publisher. The navigation stack should be subscribed to this topic and will␣
     ↪receive the goal, plan a path, and execute it to navigate the robot to the␣
     ↪target location.
```