Homework 2

Robotics 2026 - ShanghaiTech University

1 Robots. (5 %)

1. Make a list: Name the five coolest/ best/ most interesting robots (or types of robots) for **you**. Those have to be robots that actually exist(ed) and work(ed). (As long as you name five robots this answer is correct - it is up to you what robots you find interesting!)
2. For one of those robots, write at least three sentences **why** this is a cool robot.
3. Imagine a humanoid household robot. This robot should do lot's of things in the house, for example wash the dishes and put them back in the shelf, cook dinner, play with your kids, walk the dog, go shopping, etc. What do you think - what are the greatest challenges towards humanoid household robots? Name three to five challenges, max two sentences each.

Save your answers in a text file called "Robots.txt" in the root folder of your HW2 repo (see below).

2 ROS Tutorials (5%)

Work through the tutorials in chapter Beginner: CLI tools and Beginner: Client Libraries till **Writing a simple publisher and subscriber (C++)**.

3 Academic Honesty

Please everybody do the homework completely by yourself. You are doing this to learn about robotics and ROS. Do not ask other students for help. Ask in piazza! If the problem persists ask TA or Prof. Schwertfeger - we will help you!

4 Preparation

You are required to follow these steps and use the framework as a remote repository. Otherwise you'll loose points in the next task!
Make a new ROS2 workspace, e.g. hw2_ws:

```
mkdir -p ~/hw2_ws/src
cd ~/hw2_ws/src
```

Clone your HW repo from gitlab...
In the repository add a remote repo that contains the provided HW2 framework files:
cd into your repo

```
git remote add HW2frame http://robotics.shanghaitech.edu.cn/gitlab/robotics2026/public/
hw2_frame.git
```

Go and fetch the files:

```
git fetch HW2frame
```

Now merge those files with your master branch:

```
git merge HW2frame/main --allow-unrelated-histories
```

Now you should have the files in folder `HW2` in your master repo. If you do ls you should see the following folders:

```
mobile_robotics_odom mobile_robotics_interfaces jackal_msgs figure8_feedback pose2d_to_
3d
```

The rest of the git commands work as usual.

Now go and make the workspace:

```
cd ~/hw2_ws
colcon build
```

Do not forget to always source "setup.bash" in the "install" folder of your workspace for every new shell that you open!

```
source install/setup.bash
```

If everything went well up to here (i.e. no errors during compilation) you are ready to proceed with the actual homework. If there was a problem, try your best to solve it - maybe ask on piazza.

5 Calculate Odometry (30%)

Your first task is to calculate the odometry of the robot (i.e. how much did the robot move in x, y, theta since the last message). Input is the feedback data from the Jackal robot, replayed on the rosbag "`figure8_feedback`" on the topic "feedback".

You may show the fields of the feedback message by execute:

```
ros2 interface show jackal_msgs/msg/Feedback
```

In the feedback message you find info from two drives. Use this documentation to find out which one is for which wheel. As documented here, the message gives you info about the `measured_velocity` and `measured_travel` in according units. For this task it should be possible to use either of these. Prof. tested it with using `measured_travel` - that one works....

As we learned in the lectures, you'll need some data about the robot locomotion configuration in your equations. ROS (and you) finds these values in the URDF file of the robot, which describes the physical parameters and frames of the robot. "track" is the distance between the centers of the left and right wheel. As always in ROS, all distance units are in meter.

The Jackal robot does not have good odometry, because when turning wheels always have to slide over the ground (skid steer). For this reason there needs to be a factor applied to compensate for this. This is done by artificially extending the value for the distance between the wheels (because turning is harder). The factor used by Clearpath is 1.5. See the `wheel_separation_multiplier` in this page and an explanation in yet another page. Use that factor in your code! Nevertheless, the odometry you will get will not be perfect. After finishing all tasks of this HW you can see the path of the robot in rviz. For a correct solution the robot start- and end-pose from the bagfile should be less than about 40cm apart.

Program code that calculates the odometry by hand.

Output are the 2D odometry estimates you calculated, published on topic "/odom". The odometry are of message type "`mobile_robotics_interfaces/msg/Transform2DStamped`", that can be found in the downloaded code. Looking at Transform2DStamped.msg you will see that it consists of several fields along with a "Header". You can find out more about the messages by execute:

```
ros2 interface show mobile_robotics_interfaces/msg/Transform2DStamped
```

Make sure to publish the odometry with the correct timestmap of the received feedback message.

File "mobile_robotics_odom/src/odometry.cpp" has everything you need already prepared - you just need to do the calculation and fill the corresponding message! You can add this around line 48.

Compile by going into the root of your workspace and execute `colcon build`.

Test by running: `ros2 run mobile_robotics_odom odometry`

Create a rosbag named "odom" from the content of the **whole** "figure8_feedback". So you have to make sure that you start your node first, then start to record "odom", and only then start to play "figure8_feedback"!

Important: For at least every 2nd line of code that you write, add one line of comments (starting with "//"), describing what the line of code does.

6 Calculate Global Pose Estimate (30%)

Your next task is to calculate the global pose estimate of a robot in 2D. Input are odometry (2D transform) from the previous task, published on topic "/odom". The odometry are of message type "`mobile_robotics_interfaces/msg/Transform2DStamped`" while the pose are of message "`mobile_robotics_interfaces/msg/Pose2DStamped`".
Your task is to use 2D transform to calculate an estimate of the global position of the robot based on the odometry input (so chain the odometry together). Your global reference frame should have the name "odom", since we are estimating pose using odometry. The first odometry estimate received should be relative to this global frame.

Please do the calculation by hand (so use, for example, *sin*). Do not use a library like Eigen or tf for it.

Make sure to publish the pose with the correct frame_id and the timestmap of the received odometry message.
For this task create a new file called "pose.cpp" in the same folder as "odometry.cpp". Then edit "CMakeLists.txt" to add a new executable called "`pose`". Add "pose.cpp" to git!
This node is subscribing to "odom" and publishing messages of type "`mobile_robotics_interfaces/msg/Pose2DStamped`" on the topic "pose". Obviously you'll need to run the node from the previous task (or play its recorded bagfile) in order to test this node.
Important: For at least every 2nd line of code that you write, add one line of comments (starting with "//"), describing what the line of code does.
Test by running:

```
ros2 run mobile_robotics_odom pose
```

Create a bag named "pose" with all messages, as before.

7 Calculate and Publish Speed (20%)

Additionally to the pose, calculate the speed of the robot in meter per second. Publish the speed of type "`mobile_robotics_interfaces/msg/SpeedStamped`" under the topic name "/speed". Add all the needed code to "pose.cpp". Record a bag called "speed", containing all needed data (so number of odometry messages -1, staring with the 2nd).

8 Display Pose and Path in Rviz (10%)

Rviz is a ROS program to display robot data. It does not handle our 2D pose message, so a converter to 3D pose and path message is provided. Run: `ros2 run pose2d_to_3d pose2D_to_3D_node`
This will publish a "`geometry_msgs/msg/PoseStamped`" and a "`nav_msgs/msg/Path`" on the topics "pose3D" and "path3D", respectively. It listens to the output of your program on "pose".
You can run your program live together with the "pose2D_to_3D_node" or you can play your recorded "pose".
Start rviz by excuting: `rviz2`
Add a path display and a pose display and set the according topics in those displays. You also need to set the Fixed Frame in Global Options on the left to "odom".
Show the complete path and the last pose in rviz, using "odom" as fixed frame, and make screen shot. Make sure to also show (parts of) the relevant terminals in that screen shot. Save the screen shot in your repo under the file name "rviz.png".

9 Submission

Your submission consists of the framework with your changes, the bagfiles and the image, all committed to git and pushed to the gitlab server. Everything should be in folder HW2 you got by merging with the framework!

Make sure that your bagfiles only contain the topic needed. Make sure that the size of the bagfile is not bigger than needed!

The following files are important:

- ”Robots.txt” from the first task.
- Everything that was in the original ”mobile_robotics_odom/” folder - The code must compile and be the code you used to create the bagfile!
- Especially ”odometry.cpp”, including your code and the comments.
- Also ”pose.cpp” that you created.
- ”odom” bag created by you!
- ”pose” bag created by you!
- ”speed” bag created by you!
- ”rviz.png”: the screen-shot from rviz.

All non-code files created by you (so Robots.txt, rviz.png and all the bags) should be placed in the HW2 folder of the repo.