1. Differential Drive Mobile Robot Labs

1.1 Lab 4: Map Based EKF Localization

This exercise tackles the problem of the localization of the Differential Drive Mobile Robot using an a priory known map of point features. The features are represented in the map in Cartesian coordinates but they are observed by the robot in Polar Coordinates.

1.1.1 Cloning the PR_LAB4 Repository

In this section, we'll guide you through the process of cloning the "PR_LAB4" repository from GitHub onto your computer. This repository contains materials related to a laboratory exercise or project. Make sure you have Git installed on your computer before proceeding.
Follow these steps to clone the repository:

1. **Open a Terminal or Command Prompt**: Depending on your operating system (Windows, macOS, or Linux), open a terminal or command prompt.
2. **Navigate to the Desired Directory**: Use the 'cd' command to navigate to the directory where you want to store the repository. For example:
   ```
   $ cd Documents
   ```
3. **Clone the Repository**: Run the following command to clone the "PR_LAB1" repository from GitHub. Replace '<your-username>' with your GitHub username:
   ```
   $ git clone https://github.com/IFROS-MIRS/PR_LAB4.git
   ```
   If you have GitHub authentication set up, you won't need to provide credentials. If not, GitHub may prompt you to enter your username and password.
4. **Verify Cloning**: Once the cloning process is complete, navigate to the cloned repository's directory using the 'cd' command:
   ```
   $ cd PR_LAB4
   ```
   You should now be inside the cloned repository. You can check the contents to verify that everything has been cloned correctly.

That's it! You've successfully cloned the "PR_LAB4" repository onto your computer. You can now work on the contents of the repository or use them for your laboratory exercise or project.

1.1.2 Updating with your solution of LAB1

Once you have cloned the repository, you need to update the python files that you already programmed during lab1. This include the files:

1. *DifferentialDriveSimulatedRobot.py*
2. *DR_3DOFDifferentialDrive.py*
3. The *Pose3D.py* has been replaced by a new file named *Pose3D.py*. It defines a base class named *Pose* from where the class *Pose3D* inherits. You need to update the methods of the *Pose3D* class with the contents you already programmed during LAB1. You will need to change the *from Pose3D import ** of *DR_3DOFDifferentialDrive.py* file by an *from Pose import **.

Next you have to update certain variables within your files:

1. self.yaw_reading_frequency=1 (DifferentialDriveSimulatedRobot.py)
2. self.visualizationInterval=1 (SimulatedRobot.py)
3. self.vehicleIcon=VehicleIcon('DifferentialDrive.png', scale=1, rotation=90) (SimulatedRobot.py)

Now we are ready to solve the first part of the lab.

1.1.3 Part I: Localization using Dead Reckoning and a Compass

The goal of part I is to implement an Extended Kalman Filter (EKF) Localization method using the robot odometry displacement as the motion model and making updates with the compass readings. You need to follow the next steps:

1. Read the documentation of the **GFLocalization class**, check its code and complete the methods labelled *# To Do: To be completed by the Student*.
2. Read the documentation of the **EKF class**, check its code and complete the methods labelled *# To Do: To be completed by the Student*.
3. Read the documentation of the **EKF_3DOFDifferentialDriveInputDisplacement class**, and complete its methods labelled *# To Do: to be completed by the student*.
4. To **test the EKF filter**, you can program different values for the attribute *yaw_reading_frequency* of the *DifferentialDriveSimulatedRobot* class:
   (a) Using a value greater than the number of *ksteps* (number of simulation iterations), will avoid the compass updates.
   (b) Using a value equal to 1, will use the compass updates at each simulation step.
   (c) Using a value equal to 100, will use a compass update every 10 seconds.

   Comment about the results obtained.
5. **Optional: Constant Velocity Motion Model with encoder and compass updates**
   (a) Read the documentation of the **EKF_3DOFDifferentialDriveCtVelocity class**, in the file *EKF_3DOFDifferentialDriveCtVelocity.py* and complete its methods labelled *# To Do: To be completed by the Student*.
   (b) To **test the EKF filter**, in the same conditions of the previous one.

1.1.4 Part II: Map Feature Suport

Follow the next steps:

1. **Feature class**: Base class used for Feature definition. All features must inherit from this one. It defines the interface of a Feature.
   (a) Read the documentation of the **Feature class**.
   (b) Check its already provided code in the file *Feature.py*.

2. **CartessianFeature class**: Implementation of a Cartesian Feature. Inherits from the *Feature* class and provides the implementation of the pose-feature compounding operation $\boxplus$ and its related jacobians $J_{1\boxplus}$ and $J_{2\boxplus}$. It also includes the methods *ToCartesian()* and *J_2c()* which are used to be able to plot the features.
   (a) Read the documentation of the **CartesianFeature class**.
   (b) Complete the missing code labelled *# To Do: To be completed by the Student*.
3. **MapFeature class**: This class provides the functionality required to use features for map based localization.
   (a) Read the documentation of the **MapFeature class**.
   (b) Check its code in the file *MapFeature.py* and complete the requested methods.
4. **Cartesian2DMapFeature class**: This class inherits from the *MapFeature* class and implements a 2D Cartesian feature model for the Map Based Localization (MBL) problem. The Cartesian coordinates are used for both, observing as well as storing it within the map.
   (a) Read the documentation of the **Cartesian2DMapFeature class**.
   (b) Check its code in the file *MapFeature.py* and complete the requested methods.
5. **Optional:**
   (a) **2D Feature Stored in Cartesian space but observed in Polar space.**
      i. Read the documentation of the ***Cartesian2DStoredPolarObservedMapFeature class***.
      ii. Check its code in the file *MapFeature.py* and complete the requested methods.
   (b) **Create a *PolarFeature class*** using features observed and stored in Polar Coordinates.

1.1.5 Part III: Map Based EKF Localization

Follow the next steps:

1. **FEKFMBL class**: This class extends the *GFLocalization* (providing the basic functionality of a localization algorithm) and *MapFeature* (providing the basic functionality required to use features) by adding functionality to use a map based on features.
   (a) Read the documentation of the **FEKFMBL class**.
   (b) Check its code in the file *FEKFMBL.py* and complete the requested methods.
2. **The MBL_3DOFDDInputDisplacementMM_2DCartesianFeatureOM class**: This class implements a Feature EKF Map based Localization of a 3 DOF Differential Drive Mobile Robot using a 2D Cartesian feature map and an input displacement motion model .
   (a) Read the documentation of the **MBL_3DOFDDInputDisplacementMM_2DCartesianFeatureOM class**.
   (b) Check its code in the file *MBL_3DOFDDInputDisplacementMM_2DCartesianFeatureOM.py* and complete the requested methods.
3. **Test the Localization algorithm**, in the following conditions:
   (a) Without compass updates neither feature updates. To achieve this you can play with attribute s *yaw_reading_frequency* and *xy_feature_reading_frequency* .
   (b) Without compass updates but including feature updates every 50 simulation steps. To achieve this you can play with attributes *yaw_reading_frequency* and *xy_feature_reading_frequency*.

   Comment of the result obtained.
4. **Optional**
   (a) **2D Features Stored in Cartesian Space but Observed in Polar Space**
      i. Read the documentation of the **MBL_3DOFDDInputDisplacementMM_2DCartesianFeaturePolarO class**.

ii. Check its code in the file *MBL_3DOFDDInputDisplacementMM_2DCartesianFeaturePolarObservedOM.*
and complete the requested methods.
iii. **Test** the method in the same conditions than the previous algorithms.

(b) **Polar Features**: Implement the code required to run the file *MBL_3DOFDDInputDisplacementMM_2DPolarF*
which implements a Map Based EKF Localization of a Differential Drive Mobile robot using an input displacement motion model and uses Polar features observed and stored in Polar coordinates. **Test** the method in the same conditions than the previous algorithms.

(c) **Constant Velocity Motion Model**
i. Read the documentation of the **MBL_3DOFDDCtVelocityMM_2DCartesianFeatureOM class**.
ii. Check its code in the file *MBL_3DOFDDCtVelocityMM_2DCartesianFeatureOM.py* and complete the requested methods.
iii. **Test** the method in the same conditions than the previous algorithms.

Bibliography

[1] Pere Ridao and Roger Pi. *prpy: Probabilistic Robot Localization Python Library*. October 2023. Available in pdf in the ./docs/latex folder of the lab repository and in html in the `file: ./docs/html/index.html` file.