Milestone 2: Simultaneous Localization And Mapping (SLAM)

Setting Up

Server

- After connecting to the Pi through PuTTY, `cd` into `ECE4078_MY_Lab_2022` and type `git pull` to get updates to the server/robot codes.
- Charge the batteries of the Alphabot.

Client

- Install a few packages by running the command:
  ```
  pip install machinevision-toolbox-python spatialmath-python==0.8.9 matplotlib
  ```
- To confirm the required packages are installed, in a terminal type `python`, which opens python in command line, then type:
  ```
  import cv2
  from cv2 import aruco
  from machinevisiontoolbox import Image, CentralCamera
  ```
  If there is no error then everything is successfully installed and you can exit python by typing Ctrl+Z or exit().
  If you encounter the error “cannot import name aruco from cv2”, run:
  ```
  pip uninstall opencv-python opencv-contrib-python
  pip install opencv-contrib-python
  ```
- Download `milestone2.zip` and move the content to your working directory.
- Replace the keyboard control section in `operate.py` with the code you developed for M1.

Activities

Calibration (Week 3)

Wheel calibration

In the `calibration` folder, please complete `wheel_calibration.py` by filling in **line 46** for computing the scale parameter, and **line 89** for computing the baseline parameter. Run the wheel calibration script using the command `python wheel_calibration.py --ip IPADDRESS --port 8000` . This script will set the robot driving forward, and then spinning at various velocities, for durations that you specify, in order to compute the scale and baseline parameters for wheel calibration.

You can mark a 1m long straight line with masking tape on the floor, and use it as a guide to check if the robot has traveled 1m (do not have to be exact). As Alphabot2-Pi does not have wheel encoders, it has trouble traveling straight due to wheel slip. To obtain the best calibration possible, you can perform some changes such as cleaning the tyres/tracks, using speed range that your pibot usually operates (line 18 and line 61), using smaller range, or using distance less than 1m.

Camera calibration

Run the `calib_pic.py` script with the `--ip` and `--port` arguments, and then press ENTER to take a photo of the calibration rig. Position the robot such that the 8 dots are clearly visible. The photo should look something like this:

Once you have taken the photo (it will be saved as `calib_0.png`), you will now need to calibrate the camera parameters by running the command `python camera_calibration.py` . This will require you to select 8 key points from the calibration photo you just took. Select the 8 key points following the ordering shown in `calibration-fixture.png` by left clicking on each point (right click to cancel a selected point). Once all 8 points are selected, close the figure window to compute the camera matrix. This will update the camera's intrinsic parameters. You can find all calibrated parameters in the `param` folder.

SLAM (Week 4-5)

`operate.py` makes use of the camera and wheels' calibrated parameters and the SLAM components to produce a map saved as "slam.txt" in the lab_output folder. This SLAM map contains a list of identified ARUCO makers, their locations and the covariances of the estimation. Note: remember to replace the keyboard control section with your codes from M1.

ARUCO markers are square fiducial markers introduced by Rafael Muñoz and Sergio Garrido. OpenCV contains a trained function that detects the ARUCO markers, which will be used in this project (the dictionary we used to generate the markers was `cv2.aruco.DICT_4X4_100`). Alphabot will be using these ARUCO markers as road-signs to help it map the environment and locate itself.

SLAM computes the locations of the ARUCO markers using both camera-based estimation (`slam/aruco_detector.py`) and motion model-based estimation (`slam/robot.py`). Please complete **`robot.py`** by filling in the computation of the **derivatives** and **covariance** of the motion model. Please also complete **`ekf.py`** by filling in the computation of the **predicted robot state** and the **updated robot state** to finish the extended Kalman filter function.

Once `robot.py` and `ekf.py` are completed, you can test the performance of your SLAM by running `python operate.py`.

Marking

To be announced.

FAQ M2

- The scale and baseline parameters are used in robot.py. Study how they are used in the codes to help you understand how to compute them.
- Take a close look at the units of the expected output when formulating your calculations. Referring to these equations may be helpful:

  $$\dot{x} = r\omega \qquad \dot{\theta} = \frac{r}{L}(\omega_r - \omega_l)$$

- Beware of the sign error that can happen with calculating the difference between the measurement and estimate in the correction step
- The state vector x will be appended with the aruco-marker measurements. Take a note of the location of x that should be updated in the motion model.
- In the prediction step, we should update the mean belief by driving the robot.