COMP0242 Final Project: Extended Kalman Filter and Model Predictive Control

October 27, 2024 (Last Update: 20241027-1452)

1 Introduction

Consider the scenario illustrated in Figure 1 — a wheeled mobile robot operates in a factory. The robot's purpose is to transport material across the factory floor. This requires the robot to have two capabilities: it must know where it is in the factory (localize), and it must be able to control its movement to its destination. The layout of the factory is known, and the environment is populated by a set of landmarks or features that are detected by sensors mounted on the robot.

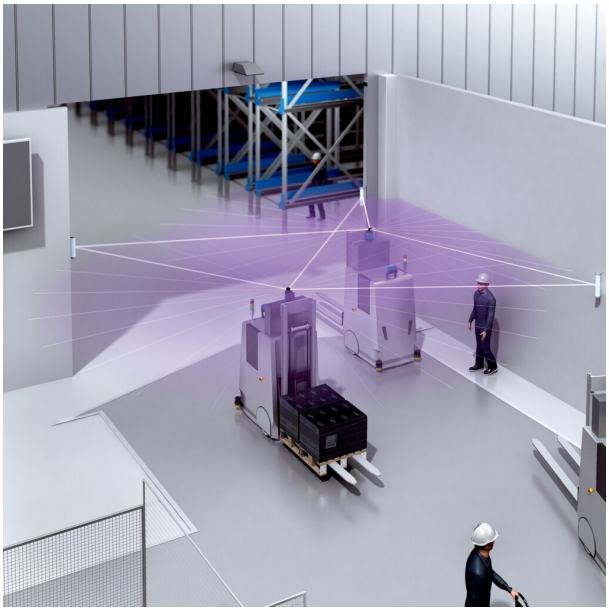

Figure 1: Illustration of a wheeled mobile robot using a laser-based ranging system. From "Navigation in the Warehouse with Navigation Scanners" (SICK Optic Electronic, S.A.U.).

This project encompasses three main themes:

1. **Apply an Extended Kalman Filter (EKF)-based estimation algorithm to estimate the pose (position and orientation of the robot).**

2. **Develop a Unconstrained MPC Regulator to work with the pybullet simulation for the robot.**
3. **Combine the EKF and the MPC to develop a system to control the trajectory of the robot in simulation.**

The maths describing the system are in Appendix A. This is the same system that is presented as the worked use case in Lecture 11 of COMP0242.

2 Prerequisites

To complete this lab, you will need to have already installed the following:

- **The latest version of RoboEnv2 from the github page (2024/10/27).**
- **The latest version of the public assignments from the GitHub page (2024/10/27). In particular, the `finals_master` directory.**
- **The model `summit_xl_description` directory and the config file called `robotnik.json` should located where they can be found.**

3 Tasks

Task 1: Extend the EKF-Based Localization System to use Range-Bearing Sensors (40%)

You will initially work with the file `robot_localization_system.py`. This file contains an EKF-based implementation of a localization system using range-only sensors. To test this system, the file `standalone_localization_tester.py` contains a lightweight simulator class (`Simulator`).

For this task, a default controller is provided in the standalone simulator, and nothing is needed from Task 2.

Activities

1. **The localization system must estimate the robot's position when the standard deviation (2σ) is less than 10cm for every timestep in the run. By analysing the error plots generated by this code, show that the initial configuration is unable to meet this requirement and discuss why you might think this is the case.**
2. **One proposed solution is to simply use more landmarks. By investigating some combinations of landmarks (e.g., distributed in a regular grid), can you identify a configuration that is able to meet the requirements? Describe what grid was required and provide evidence to show that the performance criteria have now been met.**
3. **A second proposed solution is to change the sensing system on the robot. Rather than have sensors that just return the range of the landmark to the robot, have a sensor that returns the range and bearing to the landmarks. Modify the simulator to generate range-bearing measurements, and modify the Kalman filter to process the range-bearing measurements.**
4. **Investigate again the configuration of the landmarks and identify a configuration that meets the performance requirements. Describe this configuration and show the results.**

Hints

- **Rather than edit the existing methods, write a new method for the simulator to generate the range bearing measurements, and write a new method for the estimator to update with those measurements.**
- **When dealing with angular measurements, angle wrapping is critical. Check how it is handled in the existing code and be sure to use it when computing the innovations for the filter. If you don't, the results will be spectacular.**

Task 2: Replace the Fixed Controller with an Unconstrained MPC Regulator (40%)

In this task, you will work with the full simulator. Robot localization is handled by an external "ground truth" system which provides perfect measurements of the robot state. Therefore, you do not need to use any of your answers from Task 1 to complete this task.

The provided code uses a fixed controller which always yields constant outputs. As a result, the robot will never reach its destination. In this task, you will replace the fixed controller with an unconstrained MPC regulator. This regulator was first introduced in lab session 2. We use the same classes from that lab session, but you will need to adapt them to the robotic system here. The relevant maths for the MPC are in Appendix B.

Configuration of `robotnik.json`

Before starting, pleasure ensure that the following settings are applied:

```
"init_link_base_position": [[2.0, 3.0, 0.0]],
"init_link_base_orientation": [[0,0,0.3827,0.9239]],
"motor_damping":[0],
"motor_elastic_torque": [0],
"noise_flag":[0]
```

Activities

1. **Select the values for the matrices A and B for the system. To complete this task, you will need to complete the implementation of**

   `updateSystemMatrices(self,sim,cur_x,cur_u)`

   and follow the instructions to build the linearized and discretize A, and B matrices. `cur_x` **and** `cur_u` **are, respectively, the state and control around which you are discretizing the system dynamics. For example, if you want to discretize the system at the origin of the state space then use** `cur_x=[0,0,0]` **and** `cur_u=[0,0]`**.**

   For this problem, you have two options:

   - **Linearize the system around the** `goal_state=(0,0,0)` **and the** `goal_action=(0,0)` **and never update it again**
   - **Linearize the system around the most recent**

     $$(x, u)$$

     state-control pair and update A and B at each time step

Show which of these choices is more appropriate by assessing how well they can be used to predict the future trajectory of the robot. You can achieve this by using the existing fixed controller, set to produce very small outputs and running the simulator and the linear model, and comparing the trajectory of both. To make the comparison fair, the initial states of the two models should also be the same, and close to the origin.

2. **Select a choice of the prediction horizon N and the weight matrices Q and R such that the MPC can stabilize the system. Explain your choice and For your chosen parameters, how stable is the MPC to small changes in Q and R? Update Q and R you can use the function called**

```
setCostMatrices(self, Qcoeff, Rcoeff)
```

and specify Qcoeff and Rcoeff as the gains of the matrices. If Qcoeff and Rcoeff are scalar you will use the same value for all state and control dimensions. If Qcoeff and Rcoeff are np.array they will be used to specify the weights on the main diagonal of R and Q

3. **MPCs with a large prediction horizon can be computationally very expensive and cannot sustain the update rates required for smooth and stable control. One way to address this is to add a terminal weight matrix P. Using the approach described in Lecture 04, compute a value for this weight matrix. To what degree can you reduce the prediction horizon and still have a stable controller?**

Hints

- **if the motion of the robot is too jerky you can reduce the gain to command to control the wheel rotation velocity by changing it in the configuration file: "servo_vel_gains": [3]**
- **you can control the simulation slippage by changing in line 71 of differential_drive.py the floor friction. If you increase the friction the robot will adhere more to the ground but some slippage can help the rotation of the robot**
- **Remember that the MPC regulator cost function is defined in a way that will push the robot to the zero state**
- **Think about why we have to make the input and state close to the origin when we verify A and B.**
- **To stabilize the system without terminal cost, we can choose a relatively large prediction horizon N. Please refer to slides 19-23 of lecture 04.**
- **The computation of the terminal weighting P can refer to slide 42 of Lecture 04.**
- **Think about the relationship between the EKF and MPC. Are there terms that perform a similar role for each algorithm?**
- **Since the kinematics are nonlinear and in the continuous-time domain, you have to find a linearization of the model in the (0,0,0) state first and find its discrete-time model. See slides 40–42 of Lecture 02 for more details.**

Task 3: Integrate the EKF into the Full Simulator for the Differential Drive Robot (10%)

This task builds off of the work you did in Task 1. There, the EKF was implemented using a small standalone simulator, not the full simulator used in Task 2. In this task, you will undertake the integration to get the EKF working with the full robot simulator. You will be working with the code in the file `differential_drive.py` **in the** `finals_master` **subdirectory.**

Configuration of `robotnik.json`

Before starting, please ensure that the following settings are applied:

```
"noise_flag":[1],
"robot_noise":[
    {
        "joint_cov": 0.0,
        "joint_vel_cov": 0.0,
        "joint_acc_cov": 0.0,
        "base_pos_cov": 1.0,
        "base_ori_cov": 1.0,
        "base_lin_vel_cov": 0.0,
        "base_ang_vel_cov": 0.0,
        "joint_torque_cov":0.0
    }
]
```

Activities

1. **Integrate the EKF into the simulator, ensuring it receives the appropriate control inputs, measurements and timesteps.**
2. **Validate the EKF by comparing the estimated state with the true state in the simulator. You can recall the real state of the robot by using these functions.**

Hints

1. **Check the output of the MPC to see if they are or aren't in the form the Kalman filter is expecting.**

Task 4: Compare Robot Performance With and Without the Kalman Filter (10%)

In this final task, you will compare the performance of the MPC regulator with ground truth (MPCT, which you completed in Task 2) and with the Kalman filter (MPCK, which you completed in Task 3).

Activities

1. **Design a test regime for comparing the MPCT and MPCK. This regime consists of a set of initial and target positions that you can use to evaluate the trajectory of the robot. Present your regime and describe the rationale for your choice.**

2. **Analyze and compare the performance of the MPCT and MPCK. Your analysis should be both quantitative as qualitative:**
   - **Steady-state error in reaching the desired final point.**
   - **Settling time.**
   - **Overshoot.**
3. **Plot the trajectories and error curves for both cases.**

Hint

- **You can think of a parallel here between training, validation, and test sets in machine learning. For Tasks 1–3 you have been developing datasets and using them to test and refine your algorithms. For this task you will present results on trajectories you haven't presented before.**

A Robot Equations

A.1 State Space Model

The state of the robot is its position and orientation in world-fixed coordinates,

$$\vec{x}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix}. \tag{1}$$

A.2 Process Model

The control input is the linear speed and angular velocity of the robot:

$$\vec{u}(t) = \begin{bmatrix} s(t) \\ \omega(t) \end{bmatrix}. \tag{2}$$

The process model is

$$\begin{aligned} x(t+1) &= x(t) + s(t)\cos(\theta(t))\Delta t + v_x(t) \\ y(t+1) &= y(t) + s(t)\sin(\theta(t))\Delta t + v_y(t) \\ \theta(t+1) &= \theta(t) + \omega(t)\Delta t + v_\theta(t) \end{aligned} \tag{3}$$

The process noise is additive to the position and angle states,

$$\vec{v}(t) = \begin{bmatrix} v_x(t) \\ v_y(t) \\ v_\theta(t) \end{bmatrix}. \tag{4}$$

A.3 Kinematics of a Differential Drive Robot

In the full simulator, the robot has a differential drive. This means that it has two wheels (left and right) which are driven at different angular speeds, $\omega_l(t)$ and $\omega_r(t)$. The radius of each wheel is R and the baseline — the separation between them — is B.

The linear speed and angular velocity are given by

$$\begin{aligned} s(t) &= \frac{R\omega_l(t) + R\omega_r(t)}{2} \\ \omega(t) &= \frac{R\omega_l(t) - R\omega_r(t)}{2B} \end{aligned} \tag{5}$$

A.4 Range-Bearing Sensor Model

The environment is populated by a set of landmarks. The coordinate of the ith landmark is given by

$$\vec{l}^i = \begin{bmatrix} l_x^i \\ l^i \end{bmatrix}. \tag{6}$$

The observations consist of the range and bearing to a landmark i,

$$\vec{z}^i(t) = \begin{bmatrix} r^i(t) \\ \beta^i(t) \end{bmatrix}. \tag{7}$$

First, define the quantities

$$\begin{aligned} \Delta x^i(t) &= l_x^i - x(t) \\ \Delta y^i(t) &= l_y^i - y(t) \end{aligned} \tag{8}$$

The observation equations are

$$\begin{aligned} r^i(t) &= \sqrt{\left(\Delta x^i(t)\right)^2 + \left(\Delta y^i(t)\right)^2} \\ \beta^i(t) &= \arctan\left(\frac{\Delta y^i(t)}{\Delta x^i(t)}\right) - \theta(t) \end{aligned} \tag{9}$$

For the range-only sensor, only the first row of this vector is calculated and returned.

B MPC Equations

An MPC seeks to find a sequence of control inputs which minimize the computed cost over the prediction horizon N,

$$J(t) = \sum_{k=0}^{N-1} \left(\|\vec{x}_{t+k}\|_{\mathrm{Q}}^2 + \|\vec{u}_{t+k}\|_{\mathrm{R}}^2 \right) + \|\vec{x}_{t+N}\|_{\mathrm{P}}^2 \tag{10}$$

To achieve this, the MPC predicts the future behaviour of the system using the linear equations

$$\begin{aligned} \vec{x}_t &= \vec{x}(t) \\ \vec{x}_{t+k+1} &= \mathbf{A}\vec{x}_{t+k} + \mathbf{B}\vec{u}_{t+k}, k = 0, 1, \cdots, N \end{aligned} \tag{11}$$