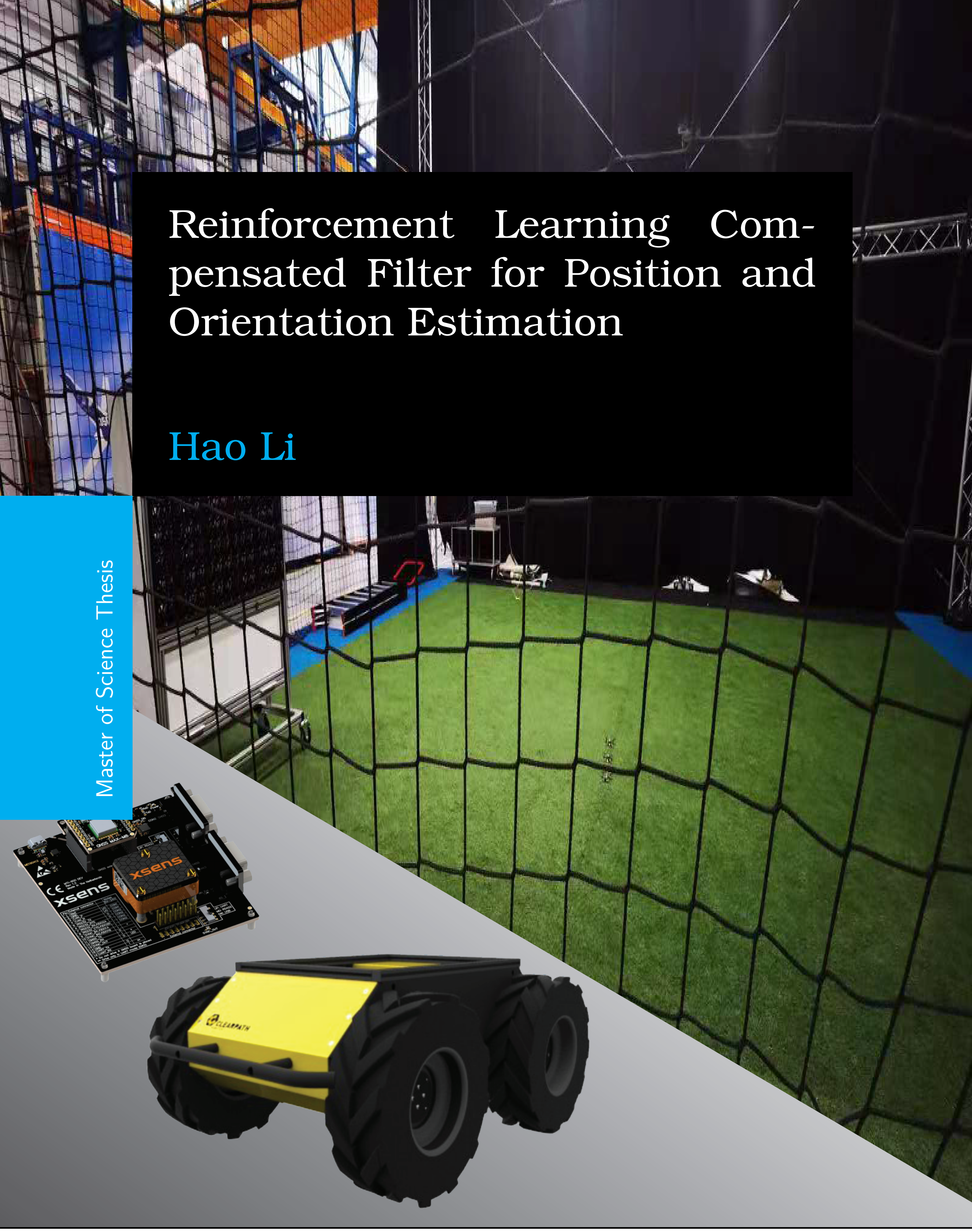


Reinforcement Learning Compensated Filter for Position and Orientation Estimation

Hao Li

Master of Science Thesis

Delft University of Technology

TUDelft Cognitive of Robotics (CoR)

Reinforcement Learning Compensated Filter for Position and Orientation Estimation

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

Hao Li

August 23, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Pose estimation provides accurate position and orientation information of the intelligent agents in real time. The accuracy of the estimation directly affects the performance of sequential tasks such as mapping, motion planning, and control. EKF (Extended Kalman Filter) is a standard theory for nonlinear pose estimation by modeling state uncertainty to Gaussian distribution. However, EKF has requirements for proper initial estimate and system noise to obtain bounded optimal estimate. Meanwhile, model nonlinearity and non-gaussian noise modeling affect the performance of EKF significantly in practical applications. In this thesis, we focus on improving the performance of nonlinear pose estimation by reinforcement learning. By formulating an EKF measurement update as a Markov Decision Process (MDP), reinforcement learning agents can be trained to learn the estimator gain through data samples and executed as the online estimator for pose estimation tasks.

Based on the above idea, we propose a novel reinforcement learning-compensated EKF estimator (RLC-EKF), where the RL agent serves as a second-time measurement update that subsides the residual error from the standard EKF estimate. The estimator is developed and testified on two specific pose estimation scenarios. Firstly, as a continuous work from the previous study, a framework for 3 DOF orientation estimation using inertial sensor and magnetometer is replicated. Then, the framework is extended by different RL algorithms training and multi-scale robustness validation. Besides, we implement the estimator on a feature-based 2D plane localization framework. The proposed framework shows the feasibility of the underlying algorithm on a localization task with a known map. As a result, the RLC-EKF estimator gives superior performance and convincible robustness compared to classical methods in severe conditions such as varying initial states, degree of noise intensities, and model covariance.

Keywords: Deep reinforcement learning; EKF; Orientation estimation; 2D Plane localization

Table of Contents

Acknowledgements **ix**

1 Introduction **1**

1-1 Background 1

1-2 Motivation 2

1-3 Research Question 3

1-4 Contribution 3

1-5 Thesis Organization 4

2 Preliminaries **5**

2-1 Extended Kalman Filter 5

2-2 Reinforcement Learning Basis 7

2-2-1 The Agent-Environment Interface 7

2-2-2 Markov Decision Process 9

2-2-3 Policy Gradient 9

2-2-4 Proximal Policy Optimization (PPO) 10

2-3 Reinforcement Learning Filter 11

2-3-1 Multiple Layer Perceptron 11

2-3-2 RL Filter Design 12

3 RL Estimator for Attitude Estimation **15**

3-1 RLC-EKF for orientation estimation 15

3-1-1 Measurements From Sensors 15

3-1-2 Orientation Estimation using EKF 16

3-1-3 Merge with RL Estimator 17

3-2 Simulation 20

3-2-1 Synthetic Data Generation 20

3-2-2 Experiment Setup 21
3-2-3 Normal Estimation Condition 22
3-2-4 Unreliable Initial Estimation 24
3-2-5 Tremendous Noise Disturbance 25
3-2-6 Generality 26
3-3 Real Dataset 28

4 RL Estimator for Localization 31
4-1 RLC-EKF for Localization 31
4-1-1 Feature-based EKF Localization 32
4-1-2 RLC-EKF Framework 34
4-2 Simulation 36
4-2-1 Synthetic Data Generation 36
4-2-2 Experiment Setup 38
4-2-3 Normal Condition 38
4-2-4 Initial Estimate 40
4-2-5 Measurement Noise 41
4-2-6 Model Covariance 41
4-3 Real Dataset 42
4-3-1 UTIAS Dataset 43
4-3-2 Dataset processing 44
4-3-3 Train and Evaluation 44

5 Conclusion 47
5-1 Question Answer 47
5-2 Discussion 48
5-3 Future works 49

A Supplementary Experiments 51
A-1 Direct RL estimator 51
A-2 Training details 52

B Code Description 55
B-1 Environment Setup 55
B-2 Folder Structure 56
B-3 Training 56
B-4 Evaluation 57

C Quaternion Basis 59
C-1 Definition 59
C-2 Quaternion operations 59
C-3 Relation Between Euler Angle and Quaternion 61

Bibliography 63

List of Figures

2-1 Scheme of kalman filtering[1] 6

2-2 Agent-environment interaction cycle 7

2-3 Visual explanation of MLP+RELU structure 12

3-1 RLC-EKF structure for 3DOF orientation estimation 18

3-2 An example showing the composition of simulated data. With $\sigma_\omega = 1 \cdot 10^{-2}, \sigma_\mathrm{a} = 5 \cdot 10^{-3}, \sigma_\mathrm{m} = 5 \cdot 10^{-3}$. for gyroscope noise $e_{\omega,t} \sim \mathcal{N}\left(0, \sigma_\omega^2 \mathcal{I}_3\right)$, accelerometer noise $e_{\mathrm{a},t} \sim \mathcal{N}\left(0, \sigma_\mathrm{a}^2 \mathcal{I}_3\right)$, and magnetometer noise $e_{\mathrm{m},t} \sim \mathcal{N}\left(0, \sigma_\mathrm{m}^2 \mathcal{I}_3\right)$ respectively. The initial state of the quaternion trajectory is randomly selected. 20

3-3 Angular velocity profile used for training 23

3-4 RMSE of the orientation estimation in normal condition 23

3-5 Comparison of RLC-EKF and EKF tracking accuracy. 4 elements in unit quaternion of reference truth(solid line), EKF estimation(dotted line) and RLC-EKF(dashed line) are displayed. The trajectories are the mean path value among 10 trials, with shaded area representing standard deviation. The cost is in negative order. . . . 24

3-6 Orientation with growing initial estimate deviation. Line type description is the same manner as section 3-2-3. 24

3-7 Orientation tracking under chaotic noise environment. The result for medium noise $1e-3$ is already given in figure 3-6. Line type description is the same manner as section 3-2-3. 25

3-8 Different angular velocity profiles used in this scenario. 26

3-9 Same profile trained from profile1 evaluating on other profiles. 27

3-10 Result visualization for profiles benchmark 27

3-11 Real dataset collection process. 28

3-12 Real data quaternion tracking result. 29

4-1 Illustration of dynamic and measurement models for ground localization task . . 33

4-2 RLC-EKF structure for Localization 34

4-3 Visualization of simulation environment 37

4-4 Qualitative result of localization task in moderate condition 39
4-5 Representative trajectories with different initial estimates for visualization 40
4-6 Qualitative results of localization with different measurement noise level. Both cost curve and variables in state vector are compared. 41
4-7 RMSE with different measurement noise level. 41
4-8 Qualitative results of localization considering different model covariance ratio, the result of ratio 1:1 is shown in Figure 4-4. 42
4-9 Hardware essentials of UTIAS dataset[2] 43
4-10 Illustration of UTIAS dataset loading and camera sensing process. In figure 4-10(a), one dataset is loaded in Matlab, we only focus on one robot and filter out others 44
4-11 Data visualization of $robot5$ from the UTIAS $dataset-1$ 45
4-12 Cost curve, trajectory tracking and variable tracking on three different evaluation scenarios. 46

A-1 Direct RL estimator for orientation estimation 51
A-2 Tracking performance of direct RL estimator. We compare the result between gyroscope update(dotted line) and RL update(dashed line). 52
A-3 Training loss of RL algorithms reading from tensorboard log. 53

List of Tables

3-1 Hyperparameters of PPO2 training process (orientation estimation) 22
3-2 Selection of action space 22
3-3 Quaternion RMSE of RLC-EKF w.r.t initial deviation 25
3-4 Advantage of RLC-EKF w.r.t different measurement nose level 26
3-5 RLC-EKF evaluating on different profiles 27
3-6 RLC-EKF on real dataset 29

4-1 Hyperparameters of PPO2 training process (localization task) 38
4-2 RMSE of the localization task 40
4-3 RMSE with different initial deviation 40
4-4 RMSE with different model covariance ratio 42
4-5 RMSE with different model covariance ratio 45

Acknowledgements

First, I would like to thank my supervisor Prof. Dr. Wei Pan. He is always supportive of providing experiment-related stuff and encourages us to enjoy the process of learning. From him, I learned to be curious about innovative ideas and never underestimate yourself because of the "impossible." Those lifetime benefits are precise for my future career life. I want to thank my daily supervisor, Ir. Yujie Tang. There are many difficulties I encountered during the thesis. She's always willing to sit with me and solve them together. She taught me how to schedule a timeline, plan experiments, which are necessary skills for a qualified researcher. This is her first year supervising students, a lot of things we have to figure out together. Now I can finally be proud to say she is an excellent tutor. I'd like to thank my family and friends for their effortless accompany during the covid-19 period. This is an exceptional time for everyone, glad that we are still facing it together. I also want to thank TU Delft for its international atmosphere, responsible staff, and incredible learning experience. I will always be proud to say I was once a TU Delfter!

Two years flicking passed by, and the bell of graduation is calling for the next splendid life journey. Let's open up a brand new chapter and step into the wonderful future.

Delft, University of Technology Hao Li
August 23, 2021

"In fact, the world needs more nerds."

— *Ben Bernanke*

Chapter 1

Introduction

1-1 Background

State estimation infers unknown states of the moving agents from the dynamic system model and noisy sensor measurements. Under the big topic of state estimation, Pose estimation focuses on estimating the robot's real-time position and orientation without containing velocity, acceleration, or adjustable factors in state vectors. Pose estimation is a general topic but can be specified in exact application scenarios due to the state of interests (SOI) required. For example, 2DOF (Degree of Freedom) position and heading angle are required for unmanned ground vehicle (UGV) localization. At the same time, the 3DOF orientation of each motion sensor placed at a person's key ankle is the most basic SOI for human motion tracking applications.

Different sensors have complementary properties; thus, they can work jointly to achieve more accurate pose estimation with sensor fusion techniques. Global Navigation Satellite System (GNSS) plays a vital role in an outdoor scenario, especially for vehicular positioning. Such absolute position provided by satellite orbiting can be fused with relative position from onboard sensors such as lidar[3] and radar[4]. In GPS-denied environments such as indoor, underwater, tunnels, INS (inertial navigation system), which utilizes IMU (inertial measurement units), is at the beginning widely researched[5]. With low-cost, high-frequency, and small-size triaxial accelerometer and gyroscope, linear acceleration and angular velocity of the moving agent can be measured respectively and then integrated with respect to the initial state to be orientation and position[6]. This classic procedure is called *dead reckoning.* Vision sensors can also provide pose information by odometry. Visual odometry (VO) reconstructs camera motions by comparing sequential images through either handcrafted features or photometric intensity. The results of image processing are then used for calculating the transformation matrix between frames. Since most VO uses a mono camera, it cannot precept pose scalability. Meanwhile, inertial odometry always suffers from accumulation drift issues. Thus Visual and inertial sensors can incorporate by either loosely-coupled or tightly-coupled methods, which is called visual-inertial odometry (VIO).

Pose estimation algorithms can also be categorized by methodologies. Traditional methods include optimization-based and filtering-based methods. Optimization-based methods are usually formulated by a maximum likelihood problem and solved by nonlinear stochastic squares (NLS). Extended Kalman Filter (EKF)[7] is one of the most renowned filtering algorithms for nonlinear pose estimation. The main idea is to linearize the system model to a differential point by first-order Taylor series approximation. State quantity modeled by Gaussian distribution is then propagated and updated the same as the standard Kalman filtering. In case of EKF fails, Unscented Kalman Filter (UKF)[8] is another substitute that utilizes unscented transformation techniques which determinedly sample some points around mean point and regenerate mean and covariance accordingly.

1-2 Motivation

The above section clarifies classic position and orientation estimation methods by both sensor fusion and methodology aspects. Those hand-designed models and algorithms work well in well-calibrated experimental conditions but are unreliable in more complex dynamics. Such a problem of lacking generalization motivates the emergence of machine learning technology applications to solve pose estimation problems. With no need to specify mathematical and physical formulas, learning methods get knowledge in a data-driven way. For supervised learning, both CNN[9] and LSTM[10] show their capabilities for pose estimation by modeling neural networks to motion dynamics or extracting universal features. However, the above methods only work in specific contexts empirically, and have no guaranteed estimation convergence in inexperienced scenario[11]. Meanwhile, as another important branch of machine learning, reinforcement learning (RL) is still in its infancy stage in the field of pose estimation. Jun[12] first proposed a novel use of reinforcement learning for estimating hidden variables and parameters of nonlinear dynamical systems. But still, the research gap between RL and pose estimation is huge and worthwhile for deeper exploration.

In this thesis, based on classical EKF, we are trying to develop an integrated reinforcement learning approach for nonlinear pose estimation. Although EKF has been widely implemented for state estimation for years, it can be sensitive due to the following factors:

- **Nonlinear complexity**: The accuracy of EKF diverges quickly or even fails in case of inappropriate model linearization.
- **Initial estimate**: EKF is only guaranteed to converge when the initial state is chosen properly nearby the true initial estimate.
- **Irregular noise**: System and measurement noise in EKF are all modeled in Gaussian distribution. In a real application, such an assumption is not always convincing. Wrong modeling of non-Gaussian noise can also severely downgrades estimation performance.
- **Calibration**: Both system model variance and noise distribution should be carefully tuned for adjusting EKF into a specific scenario.

On the other hand, deep reinforcement learning (DRL), the combination of RL and DL, learns policy in a neural network form. We aim to develop a DRL-based state estimator so the additional error brought from EKF, i.e., error because of system linearization and noise modeling, can be mitigated by learning. Through data-driven-based training, the generalization of the RL estimator can also be expanded by exploring a huge amount of different dynamic environments. So ideally, no EKF calibration is needed when implementing the well-trained estimator in a changing real world. Plus, the RL model can be trained with random initial states for every exploration episode, enhancing the estimator's robustness to enormous initial drifts.

1-3 Research Question

By specifying drawbacks of EKF and potential advantages we can get from RL in the last section, here comes our research questions:

- What's the relationship between deep reinforcement learning and filtering-based state estimator such as EKF?
- How can reinforcement learning be combined with EKF for better pose estimation performance?
- How can the proposed reinforcement learning filter idea be applied to specific pose estimation tasks such as orientation estimation or localization?

1-4 Contribution

To answer the above research questions, we formulate filtering pose estimation problems as a Markov Decision Process (MDP) to find the relationships between RL and EKF. The contributions of this thesis can be summarized as follows:

- A novel reinforcement learning compensated EKF estimator structure is presented. The estimator contains two parts. The first part is a standard EKF estimation. The estimator's second part keeps the structure as the EKF measurement model but learns the measurement gain by reinforcement learning algorithms.
- As a continuous work from the previous study, an RL filtering framework for 3DOF orientation estimation using IMU and magnetometer is successfully trained by well-known reinforcement learning algorithm PPO2[13]. Experiments are also extended by more detailed robustness validation w.r.t. different tracking curves, system noises, and initial states.
- A brand new RL filtering framework for feature-based 2D vehicle localization is built up and successfully trained and evaluated with both simulated and actual datasets.
- Successful implementations on two practical applications prove the applicability of the proposed RL estimator idea on position and orientation estimation tasks.
- Proposed idea shows more superiors tracking accuracy on both two frameworks compared with pure EKF results. We also prove that the robustness of the RL filter excels

the traditional EKF when facing more dynamic environments, such as varying noises and initial estimates. Even when changing the EKF model covariance (correspond to no EKF calibration), our RL estimator can still give acceptable results when EKF is working clumsily.

1-5 Thesis Organization

The outline of this thesis is given as follows: Chapter 2 introduces the preliminaries for the proposed research, mainly the basis of filtering-based nonlinear state estimation as well as reinforcement learning. Chapter 3 presents the methodology of the proposed RLC-EKF orientation estimation framework, followed by results from both simulation and real data experiments. With a similar paper structure, a feature-based 2D localization RC-EKF is explained in Chapter 4. At last, as the summary of the whole thesis report, Chapter 5 gives out the conclusion of the thesis work clarifying both achievements and deficiencies.

Chapter 2

Preliminaries

In this chapter, we go through the background knowledge covered in this thesis. First, the most popular Bayesian filter for pose estimation, EKF, is reviewed. Then the fundamental understanding of reinforcement learning is introduced. In the end, a RL nonlinear estimator is constructed by combining EKF's state estimation structure and RL's policy that computes the end-to-end estimator gain.

2-1 Extended Kalman Filter

Probabilistic robots acquire their states from internal actuator inputs and external sensor measurements. Compared to optimization-based estimation methods, Bayesian filtering-based methods are more computationally effective. For non-linear system, EKF is the most frequently used algorithm, although it cannot promise optimal estimation like Kalman filter does for linear system[14].

General EKF solves non-linear state estimation problem in an recursive way. The process contains two sectors: prediction and measurement update. Consider the non-linear system as follows:

$$\begin{aligned} x_{k+1} &= f\left(x_k, u_k\right) + w_k \\ y_k &= h\left(x_k\right) + v_k \end{aligned} \tag{2-1}$$

Where $x_k \in \mathbb{R}^n, y_k \in \mathbb{R}^m$ are the state vector and measurement vector at time k respectively. $w_t \sim \mathcal{N}(0, Q)$ and $v_t \sim \mathcal{N}(0, R)$ are white noise (zero mean gaussian distribution) of the process model and measurement model with correspondent covariance Q and R, respectively. $f(\cdot)$ is the process model, which transfers prior state x_k to new state x_{k+1} through current actuator input u_k. $h(\cdot)$ is the measurement model, which makes predictions about sensor measurements based on current state. Such predictions will later on be compared with real sensor measurements to refine the estimation.

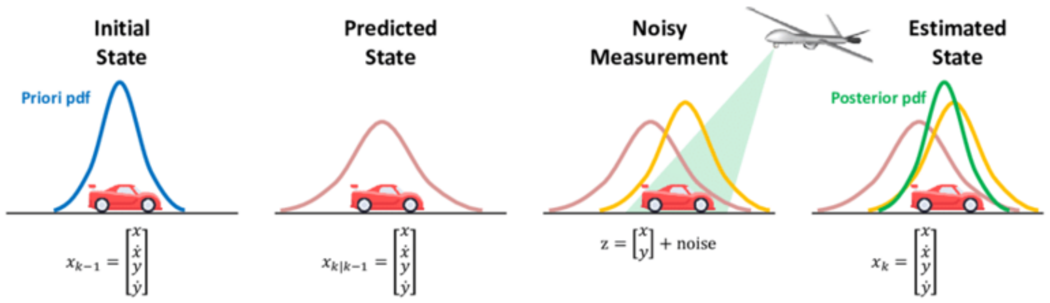

Initial State
Priori pdf
$x_{k-1} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}$
Predicted State
$x_{k|k-1} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}$
Noisy Measurement
$z = \begin{bmatrix} x \\ y \end{bmatrix} + \text{noise}$
Estimated State
Posterior pdf
$x_k = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}$

Figure 2-1: Scheme of kalman filtering[1]

In EKF, not only the noise is modeled as Gaussian, robots belief over state is also formulated in a Gaussian manner.

$$bel(x_k) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{2-2}$$

This multivariate normal distribution can be written as $x_k \sim \mathcal{N}(\mu_k, \Sigma_k)$, where μ_k and Σ_k are mean and covariance of the state at step k, respectively. Starting with the initial belief $x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$, the estimation is amended by *time update* and *measurement update* process recursively.

Time update

First, state is propagated by process model by:

$$\begin{aligned} \hat{\mu}_{k+1|k} &= f_k\left(\hat{\mu}_{k|k}, u_k\right) \\ \Sigma_{k+1|k} &= F_k \Sigma_{k|k} F_k^\top + Q_k \end{aligned} \tag{2-3}$$

Here F is linearized transfer function calculated from original non-linear form by first order Taylor expansion, we call it matrix of partial derivatives (the Jacobian). Similar we can also have Jacobian H of the measurement model, they are together given by:

$$F_k = \left.\frac{\partial f}{\partial \boldsymbol{\mu}}\right|_{\hat{\boldsymbol{\mu}}_{k|k}, \boldsymbol{u}_k}, \quad H_k = \left.\frac{\partial h}{\partial \boldsymbol{\mu}}\right|_{\hat{\boldsymbol{\mu}}_{k+1|k}} \tag{2-4}$$

Measurement update

The measurement model compares the difference between realistic sensor measurements and measurements ought to be observed from the state estimation we get from the previous prediction step. The difference between them is called *innovation*, which is then contributed to build *kalman gain* and eventually correct the estimation. The measurement model is written in:

$$\begin{aligned} \hat{\mu}_{k+1|k+1} &= \hat{\mu}_{k+1|k} + K_k \varepsilon_t \\ \Sigma_{k+1|k+1} &= (I - K_k H_k)\Sigma_{k+1|k} \end{aligned} \tag{2-5}$$

With measurement innovation ε_t, covariance innovation S_k and near-optimal kalman gain K_k given as:

$$\begin{aligned}\varepsilon_k &= y_k - h(\hat{\mu}_{k+1|k}) \\ S_k &= H_k \Sigma_{k+1|k} H_k^\top + R_k \\ K_k &= \Sigma_{k+1|k} H_k^\top S_k^{-1}\end{aligned} \tag{2-6}$$

Till now, we have the complete pipeline conducting estimation from state k to k+1. By doing this iteratively, our Kalman filter can finally start spinning. The Kalman gain K, which quantities the relative uncertainty between measurements and current state estimate, is the most critical intermediate value that affects the accuracy of our Kalman filter.

2-2 Reinforcement Learning Basis

As one research branch of three main machine learning paradigms, reinforcement learning has been widely researched for years. In the beginning, the research focus is on game and control theories, etc.. While the topic of pose estimation with RL hasn't been explored much. To extend the flexibility and generality of RL policy, deep neural network (DNN) is first time served as a policy approximation in deep Q-learning [15]. This thesis will also formulate our nonlinear estimator through a deep reinforcement learning (DRL) manner. As a prerequisite, the most fundamental philosophy of reinforcement learning is introduced in this section, part of images and formulas are from textbook [16].

2-2-1 The Agent-Environment Interface

Reinforcement learning learns from interactions between *agent* and *environment* continually to achieve a goal. At discrete time step t, the agent at state S_t takes an action $a_t \in A_t$, which will lead the agent to a new state S_{t+1}. Meanwhile, as the feedback of the conducted action, the environment gives back a reward R_t. And the aim for agent is to maximize the overall reward through the whole episode time.

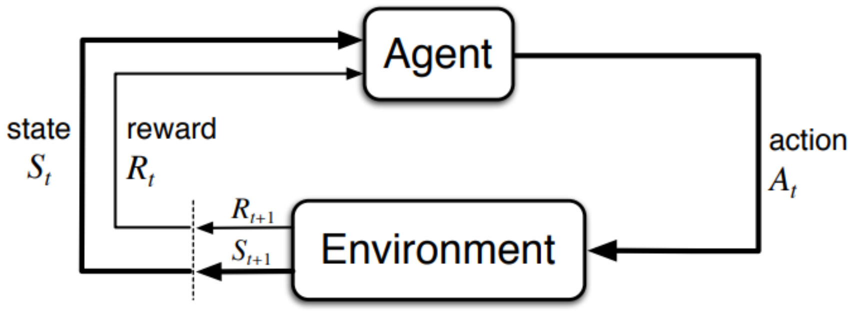


Figure 2-2: Agent-environment interaction cycle

In DRL, related components can be summarized as follows:

Agent & environment

An RL agent is the entity that learns by training to decide moves for optimal reward. The environment is where the agent lives and interacts with. It transfers the agent according to the action it takes and returns feedback. An agent only manipulates its own move and takes no adjustment to the environment.

State & observation space

A state contains the complete information of our SoI, which may include both ego agent information and other objects' information(e.g., landmarks) in the surroundings. Observation, in another word, only contains partial state information that RL problem concerns. If the domain is fully observable, observation equals to state. In DRL, state or observation space can be various forms, e.g., scalar, vector, matrix, depending on the state variables' shape.

Action space

Action space restricts the range of available actions for each step. Action can be either discrete or continuous. In pose estimation problem, it's usually continuous.

Policy function

Policy decides the most appropriate action for an agent to take based on the current state. It can be either deterministic or stochastic. Deterministic policy outputs one and only action while stochastic policy gives out probability distribution of the actions. By performing a mapping from state to a set of an action probability distribution, we have a policy function $\pi_t(s, a)$. In DRL, instead of utilizing a fixed look-up policy table such as Q-learning[17], a neural network is hired as a parameterized approximator for policy. We denote the parameters of such a policy by θ, and a stochastic policy network can be represented by:

$$a_t \sim \pi_\theta\left(\cdot \mid s_t\right) \tag{2-7}$$

Value function

We should define the value of the occurred states or actions so that optimal policy can be learned from experience. The most simplest value function would be action-space pair $r_t = R\left(s_t, a_t\right)$. However, as the goal of RL is to obtain the maximum accumulative returns, such value function cannot represent long-term benefits. The most well-used value functions in DRL are:

$$\begin{gathered} V^\pi(s) = \underset{\tau \sim \pi}{\mathrm{E}}\left[R(\tau) \mid s_0 = s\right] \\ Q^\pi(s, a) = \underset{\tau \sim \pi}{\mathrm{E}}\left[R(\tau) \mid s_0 = s, a_0 = a\right] \\ R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \end{gathered} \tag{2-8}$$

Where $R(\tau)$ is the discounted cumulative return of the whole remaining episode. $V^\pi(s)$ is the expected total reward of state s including all actions possibility while $Q^\pi(s, a)$ is the expected total reward of exact action q at state s.

2-2-2 Markov Decision Process

It is ideal to acquire batch information and optimize the policy in a Bayesian smoothing manner. However, as the agent interacts with the environment simultaneously and each step has relations with previous moves, such an optimization method is not feasible in RL. Instead, to learn incrementally, *Markov Property* is the core assumption trusted by RL through the whole story.

Markov Property

In a random process where current and all past states are given, the conditional probability distribution of the next state depends solely on current state. Markov property in mathematics can be represented as:

$$\mathbb{P}\left[S_{t+1} \mid S_t\right]=\mathbb{P}\left[S_{t+1} \mid S_1, \ldots, S_t\right] \tag{2-9}$$

Markov Decision Process (MDP) is a mathematical framework used for modeling decision-making problems, which also shares Markov property. An general MDP framework is defined by a 4-tuple form $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma>$:

$$\begin{aligned}
\mathcal{P}_{ss'}^a &= \mathbb{P}\left[S_{t+1}=s' \mid S_t=s, A_t=a\right] \\
\mathcal{R}_s^a &= \mathbb{E}\left[R_{t+1} \mid S_t=s, A_t=a\right]
\end{aligned} \tag{2-10}$$

Where $\mathcal{S}$ is the set of all valid states, $\mathcal{A}$, is the set of all valid actions. $\mathcal{P}_{ss'}^a$ is the state transition probability from s to s' when taking action a. $\mathcal{R}_s^a$ is the state reward of the action a and $\gamma \in [0,1)$ is the discount factor. Here, the next state and reward depend on what action the agent picks, thus called the decision process.

2-2-3 Policy Gradient

The goal of an RL problem is maximizing the expected return value of the whole trajectory. Suppose the state transition is stochastic, the expected return of a policy π for a trajectory T can be calculated as:

$$\begin{aligned}
P(\tau \mid \pi) &= \rho_0\left(s_0\right) \prod_{t=0}^{T-1} P\left(s_{t+1} \mid s_t, a_t\right) \pi\left(a_t \mid s_t\right) \\
J(\pi) &= \int_\tau P(\tau \mid \pi) R(\tau) = \underset{\tau \sim \pi}{\mathrm{E}}[R(\tau)]
\end{aligned} \tag{2-11}$$

Then the optimization task of a RL problems comes to find the optimal policy for satisfying:

$$\pi^*=\arg\max_\pi J(\pi) \tag{2-12}$$

By definition, action-value function Q has a direct relation with policy. So some methods are also searching the optimal Q function instead. Such methods work better in an environment with finite and discrete states. Because multiple actions on a trajectory are optimized together, there is always no optimal policy that deterministically selects an action.

$$a^*(s)=\arg\max_a Q^*(s, a) \tag{2-13}$$

Value-based RL algorithm such as deep Q-learning only applies to discrete action space, plus it's not guaranteed to converge with high state uncertainties. In this chapter, we only introduce policy-based algorithms.

Considering a parameterized policy π_θ. Parameter set θ can be optimized recursively by:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J\left(\pi_\theta\right)\big|_{\theta_k} \tag{2-14}$$

Where $\nabla_\theta J\left(\pi_\theta\right)\big|_{\theta_k}$ is the policy gradient at iteration k which is calculated by log-derivative trick:

$$\nabla_\theta J\left(\pi_\theta\right) = \underset{\tau \sim \pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right) R(\tau)\right] \tag{2-15}$$

2-2-4 Proximal Policy Optimization (PPO)

PPO shares a similar structure with A2C[18] while keeping a distinctive policy gradient style similar to Trust Region Policy Optimization (TRPO)[19] at the same time. In PPO, there are one critic and two actor networks approximating value function and policy, respectively. Motivated by policy gradient methods, PPO attempts to find the most valuable improvement possible during policy updating. Meanwhile, it sets a distance constraint between the new and old policy, so the update is neither so far nor so close. This distance is represented in a *KL-Divergence* form, which measures the difference between two probability distributions. The calculation of the KL-Divergence can be complex, so simplifications are needed. Different from TRPO, PPO simplifies KL-Divergence in a first order Taylor expansion instead of two. PPO also invites several tricks, which makes it a more straightforward and more robust algorithm.

Still in a gradient ascend manner, PPO updates the policy via:

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a \sim \pi_{\theta_k}}{\mathrm{E}}\left[L^{clip}\left(s, a, \theta_k, \theta\right)\right] \tag{2-16}$$

Where L^{clip} is the objective function give by:

$$L\left(s, a, \theta_k, \theta\right) = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s, a), g\left(\epsilon, A^{\pi_{\theta_k}}(s, a)\right)\right) \tag{2-17}$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases} \tag{2-18}$$

Here $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a)$ is the surrogate advantage which measures the relevance of the current policy π_θ towards the old policy π_{θ_k}, similar in TRPO. $A^{\pi_{\theta_k}}(s, a)$ is the estimated advantage at time t. ϵ is a small range parameter (usually 0.1-0.2) which controls the boundary of the constrain.

When estimated, advantage A of the action-state pair is positive, which means the more probability of the action, the more changes on policy. So there is a threshold of $1+\epsilon$ to avoid the policy updating so far away from the previous one. Vice versa for negative advantage, $1-\epsilon$ sets the lower bound for updating the policy.

Except for policy, the value function should also accept optimization. The objective function is the mean square error between the current state value and rewards-to-go from the exploration period. The complete process of a PPO algorithm is given as follows:

Algorithm 1 Proximal Policy Optimization (PPO)

Inputs: Initial policy parameters θ_0, initial value function parameters ϕ_0
Outputs: Trained policy and value function

```
1: for $k = 0, 1, 2, \ldots$ do
2:     Run policy $\pi_k = \pi(\theta_k)$ several steps according to the update frequency and collect set
       of trajectories $\mathcal{D}_k = \{\tau_i\}$.
3:     Compute rewards-to-go $\hat{R}_t$.
4:     Compute advantage estimates, $\hat{A}_t$ based on the current value function $V_{\phi_k}$.
5:     Update the policy by maximizing the PPO-Clip objective:
           $$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right)$$
       via stochastic gradient ascent with Adam.
6:     Fit value function by regression on mean-squared error:
           $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$
       via gradient descent algorithm.
7: end for
```

2-3 Reinforcement Learning Filter

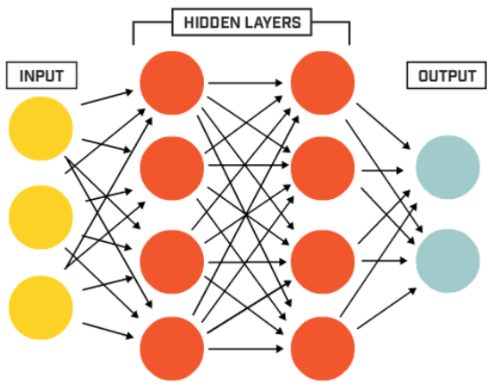
The relationship between reinforcement learning and Kalman filter should be identified to build up a Bayesian filter through DRL. In this section, EKF is firstly represented as a dynamic MDP process. Then the variables that comprise an RL environment are associated with components in EKF. In the end, the proposed RL filter is formulated in an EKF structure. The idea is first presented in [20].

2-3-1 Multiple Layer Perceptron

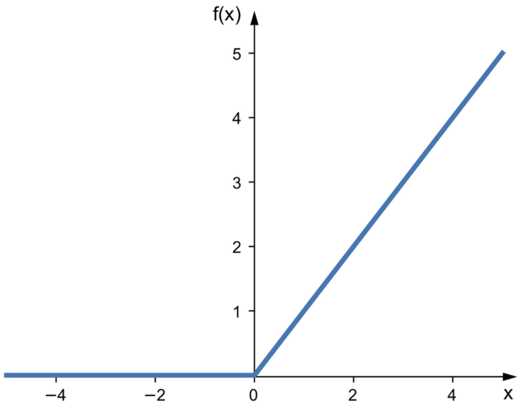
In DRL, DNNs are employed as the approximators of policy function and value function. In our PPO algorithm, actors and critics are structured by several fully connected multiple-layer perceptrons (MLP) followed by a rectified linear unit (RELU) as the activation function. This is the classical network structure in DRL. Other structures such as LSTM and CNN are not chosen because our state and action states are continuous, and no image features are included in our experiment. The most typical MLP includes three layers: input layers, hidden layers, and output layers. All neural networks are fully connected, as shown in figure 3-2(d). For policy networks, the input and output of the MLP depend on the size of the state and action vector, respectively. While the hidden layer size should be selected well based on the complexity of the filtering framework. An MLP containing two hidden layers that can be represented in mathematics as:

$$z = \mathrm{MLP}_{\mathrm{w}}^{2}(x) = \mathrm{w}_2 \left[f\left(\mathrm{w}_1 \left[f\left(\mathrm{w}_0 \left(x + b_0 \right) \right] + b_1 \right] \right) + b_2 \right] \tag{2-19}$$

where x is the input state, $[w_0, w_1, w_2]$ are weights for each layer, and f is RELU activation added after each layer.



(a) Typical MLP structure



(b) Schematic diagram of RELU function

Figure 2-3: Visual explanation of MLP+RELU structure

RELU is activation function representing in $f(x) = x^+ = \max(0, x)$, which simply filters out negative results. Activation functions always appear in pair with the neural network layer as they give advantages to the backpropagation process.

2-3-2 RL Filter Design

The final EKF update equation in 2-5 can be written into a MDP form:

$$\begin{gathered} \hat{x}_{k+1} = f\left(\hat{x}_k\right) + K_k\left(y_{k+1} - g\left(f\left(\hat{x}_k\right)\right)\right) \\ \tilde{x}_{k+1} \sim \mathcal{P}\left(\tilde{x}_{k+1} \mid \tilde{x}_k, y_{k+1}, K_k\right), \forall k \in \mathbb{Z}_+ \end{gathered} \tag{2-20}$$

where next step state only relates to current state $\hat{x_k}$, measurement y_{k+1} and kalman gain K_k.

In EKF kalman gain is computed by measurement innovation, which essentially measures the difference of the real state x and estimated state $\hat{x}$. In DRL offline training process, assume the real state x is known for each step, then the mapping from $\hat{x_k}$ to K_k can be modeled as a non-linear function. By treating the non-linear mapping function as the RL policy π and kalman gain K as the RL action, the RL state estimator in a MDP tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ is structured as follows:

- State $\hat{x_k} \in \mathcal{S}$, estimator gain $a\left(\hat{x}_k\right) \in \mathcal{C}$, system transition probability which relates to action $\mathcal{P}\left(\tilde{x}_{k+1} \mid \tilde{x}_k, a\left(\hat{x}_k\right)\right) \in \mathcal{P}$
- Reward function $R\left(\hat{x}_k, K_t\right) \in \mathcal{R}$ which measures the value of the action-state pair. In state estimation the mean square error between real and estimated state is usually utilized.
- Policy $\pi_\theta(a(\cdot) \mid \tilde{x})$ which employs MLPs approximator as the mapping from estimated state to estimator gain (action).

In conclusion, we create a DNN formed policy to learn the estimator gain within a traditional filtering framework instead of computing through handcrafted mathematical formulas. Theoretically, any RL algorithm that is able to optimize a policy can work on training the model through the proposed estimator framework. Likewise, we can apply the filtering framework on different existing physical models, to solve different estimation problems.

Chapter 3

RL Estimator for Attitude Estimation

In this chapter, we propose an RL-compensated EKF estimator (RLC-EKF) with the inspiration of the idea introduced in section 2-3-2. The estimator is applied on a 3DOF orientation estimation framework, which combines sensor measurements from inertial measurement units (IMU) and magnetometer for accurate orientation. The chapter is structured as follows. Firstly, a standard EKF orientation estimation algorithm from textbook[21] is merged with the proposed RL estimator idea to formulate the RLC-EKF framework. Secondly, the framework is trained and tested by synthetic data generated from the mathematical model. The performance and robustness are evaluated concisely w.r.t several considerations. Finally, the framework is taught and assessed on a real dataset showing its capacity in practical applications.

3-1 RLC-EKF for orientation estimation

3-1-1 Measurements From Sensors

Gyroscope measurement model

A tri-axis gyroscope measures angular velocity in x, y, z direction represented in the sensor body frame. Gyroscope measurements comprise of actual angular velocity $\omega_{nb,t}^{b}$ from body frame b to navigation frame n, time-varying gyroscope bias $\delta_{\omega,t}$ and measurement noise $e_{\omega,t}$, assuming that the navigation frame is fixed and the earth rotation is negligible.

$$\begin{aligned} y_{\omega,t} &= \omega_{\mathrm{nb},t}^{\mathrm{b}} + \delta_{\omega,t}^{\mathrm{b}} + e_{\omega,t}^{\mathrm{b}} \\ \delta_{\omega,t+1}^{\mathrm{b}} &= \delta_{\omega,t}^{\mathrm{b}} + e_{\delta_{\omega},t}^{\mathrm{b}} \end{aligned} \tag{3-1}$$

In above equation, gyroscope bias $\delta_{\omega,t}^{\mathrm{b}}$ grows with time. $e_{\omega,t}^{\mathrm{b}} \sim \mathcal{N}(0, \Sigma_{\omega}), e_{\delta_{\omega},t}^{\mathrm{b}} \sim \mathcal{N}(0, \Sigma_{\delta_{\omega},t})$ are zero mean gaussian distributions for gyroscope noise and gyroscope bias noise respectively.

Accelerometer & magnetometer measurement model

Accelerometer and magnetometer measure earth's gravity and magnetic field respectively. Their measurement models share the same mechanism which transfers the local measurements from earth frame to sensor frame.

$$\begin{aligned} y_{\mathrm{a},t} &= -R_t^{\mathrm{bn}} g^{\mathrm{n}} + e_{\mathrm{a},t}^{\mathrm{b}} \\ y_{\mathrm{m},t} &= R_t^{\mathrm{bn}} m^{\mathrm{n}} + e_{\mathrm{m},t}^{\mathrm{b}} \end{aligned} \tag{3-2}$$

Where $g^{\mathrm{n}} = \begin{pmatrix} 0 & 0 & g \end{pmatrix}^\top, m^{\mathrm{n}} = \begin{pmatrix} \cos\delta & 0 & \sin\delta \end{pmatrix}^\top$ are the local gravity and magnetic intensity vectors, δ is the geometrical dip angle of the experiment place. By comparing the magnetic field direction of the earth and measured sensor local frame, the relative sensor heading angle can be computed. This heading angle can compensate the orientations around gravity vector that accelerometer cannot provide.

3-1-2 Orientation Estimation using EKF

A typical filtering-based orientation estimation algorithm integrates angular velocity from the gyroscope as the process model. While accelerometer and magnetometer together formulate the measurement model. The purpose of sensor fusion is to utilize the gravity and local magnetic field information measured by accelerometer and magnetometer to correct integrated orientation from the gyroscope, which will improve estimation accuracy. Orientation in this chapter is presented in unit quaternion states. More computation rules about quaternion states can be found in Appendix C.

Process model

Quaternion update can be expressed by:

$$\begin{aligned} \hat{q}_{t|t-1}^{\mathrm{nb}} &= \hat{q}_{t-1|t-1}^{\mathrm{nb}} \odot \exp_{\mathrm{q}}\left(\frac{T}{2} y_{\omega,t-1}\right) \\ P_{t|t-1} &= F_{t-1} P_{t-1|t-1} F_{t-1}^\top + G_{t-1} Q G_{t-1}^\top \end{aligned} \tag{3-3}$$

where $\hat{q}^{\mathrm{nb}}$ is the relative orientation between sensor frame and earth frame represented in quaternion state, P is the covariance matrix that quantities state uncertainty. $Q = \Sigma_\omega$ as the gyroscope measurement noise. y_ω is the raw measurement from gyroscope. T is the sampling time. $\odot$ and $\exp_{\mathrm{q}}(\cdot)$ correspond to quaternion's multiplication and quaternion's exponential function, respectively. F and G are the differentials of quaternion's transfer function regarding quaternion and noise, respectively, and given by:

$$F_{t-1} = \left(\exp_{\mathrm{q}}\left(\frac{T}{2} y_{\omega,t-1}\right)\right)^{\mathrm{R}}, G_{t-1} = -\frac{T}{2}\left(\hat{q}_{t-1|t-1}^{\mathrm{nb}}\right)^{\mathrm{L}} \frac{\partial \exp_{\mathrm{q}}(e_{\omega,t-1})}{\partial e_{\omega,t-1}} \tag{3-4}$$

where $(\cdot)^L$ and $(\cdot)^R$ are the left- and right- quaternion-product matrices respectively.

Measurement model

The measurement model is defined as:

$$\tilde{q}_{t|t}^{\mathrm{nb}} = \hat{q}_{t|t-1}^{\mathrm{nb}} + K_t \varepsilon_t \tag{3-5}$$

$$\tilde{P}_{t|t} = P_{t|t-1} - K_t S_t K_t^\top \tag{3-6}$$

where $\varepsilon_t = y_t - \hat{y}_{t|t-1}$. y_t and $\hat{y}_t$ is actual and measured value of the accelerometer and magnetometer.

$$y_t = \begin{pmatrix} y_{\mathrm{a},t} \\ y_{\mathrm{m},t} \end{pmatrix}, \quad \hat{y}_{t|t-1} = \begin{pmatrix} -\hat{R}^{\mathrm{bn}}_{t|t-1} g^{\mathrm{n}} \\ \hat{R}^{\mathrm{bn}}_{t|t-1} m^{\mathrm{n}} \end{pmatrix} \tag{3-7}$$

S_t and K_t are iteratively updated from last step covariance matrix:

$$S_t = H_t P_{t|t-1} H_t^\top + R, \quad K_t = P_{t|t-1} H_t^\top S_t^{-1} \tag{3-8}$$

where R is the covariance matrix for accelerometer and magnetometer noise. H_t is the differential of both acceleration and gravity measurement function regarding quaternion given by:

$$H_t = \begin{pmatrix} -\left.\frac{\partial R^{\mathrm{bn}}_{t|t-1}}{\partial q^{\mathrm{nb}}_{t|t-1}}\right|_{q^{\mathrm{nb}}_{t|t-1} = \hat{q}^{\mathrm{nb}}_{t|t-1}} & g^{\mathrm{n}} \\ \left.\frac{\partial R^{\mathrm{bn}}_{t|t-1}}{\partial q^{\mathrm{nb}}_{t|t-1}}\right|_{q^{\mathrm{nb}}_{t|t-1} = \hat{q}^{\mathrm{nb}}_{t|t-1}} & m^{\mathrm{n}} \end{pmatrix}, \quad R = \begin{pmatrix} \Sigma_{\mathrm{a}} & 0 \\ 0 & \Sigma_{\mathrm{m}} \end{pmatrix} \tag{3-9}$$

Following above process, both quaternion's mean value and covariance are propagated and refined. But it's no longer in unit form. So the last step is for quaternion re-normalization.

$$\hat{q}^{\mathrm{nb}}_{t|t} = \frac{\tilde{q}^{\mathrm{nb}}_{t|t}}{\left\|\tilde{q}^{\mathrm{nb}}_{t|t}\right\|_2}, \quad P_{t|t} = J_t \tilde{P}_{t|t} J_t^{\mathrm{T}} \tag{3-10}$$

with,

$$J_t = \frac{1}{\left\|\tilde{q}^{\mathrm{nb}}_{t|t}\right\|_2^3} \tilde{q}^{\mathrm{nb}}_{t|t} \left(\tilde{q}^{\mathrm{nb}}_{t|t}\right)^\top \tag{3-11}$$

3-1-3 Merge with RL Estimator

Above EKF orientation estimation framework works well if the senor model noise and measurement noise are well-calibrated. However, since the non-linear function is linearized in EKF, the linearization error is inevitable. Thus the accuracy of EKF still has space to improve. Also, EKF models noise solely in Gaussian distribution, limiting its performance when noise is not regular in practice. To mitigate the above EKF's deficiencies, an RL approach to compensate EKF is proposed in this section, specifically for our orientation estimation framework. The structure of the RL estimator is already introduced in 2-3.

The high-level purpose is to let the RL estimator learn from the residual error between EKF estimation and nominal ground truth orientation during the training process. This error can then be compensated by a second-time measurement update, where the estimator gain matrix is obtained from the RL policy. The output of the RL strategy mainly depends on the current RL state, i.e., the residual orientation.

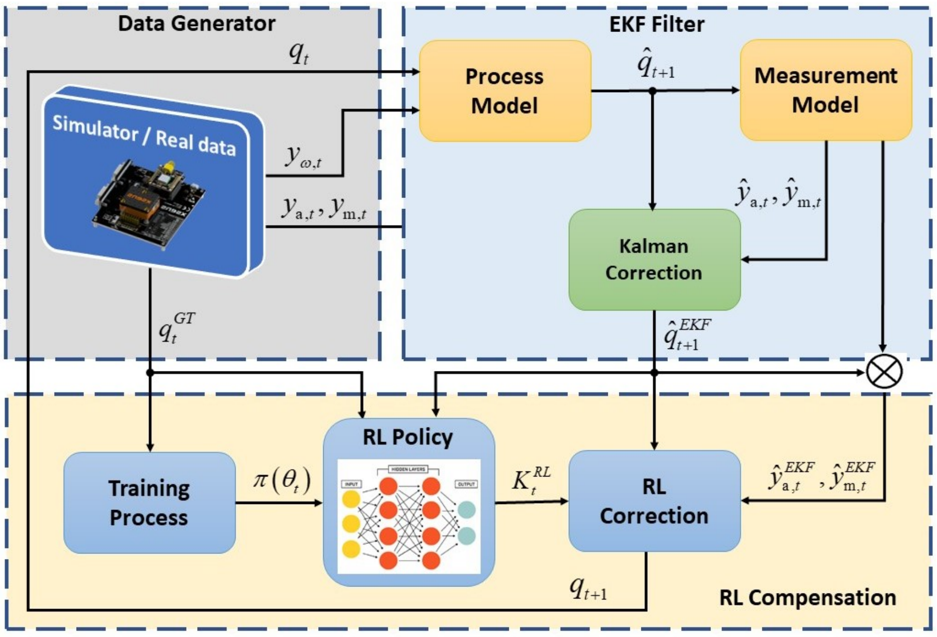


Figure 3-1: RLC-EKF structure for 3DOF orientation estimation

Due to the exact norm quantity limitation, it's not ideal to use quaternion representation as our RL state. Instead, we use the form of orientation deviation as the state vector of the RL estimator, denoted by $\hat{\eta}_t^{\mathrm{nb}}$. The learned compensation orientation will also be in a deviation form during the calculation. We can treat this as the relative rotation between two quaternions represented in matrix Lie group $SO(3)$. Since in $SO(3)$, it's also convenient to transfer the compensated orientation back to quaternion states by the exponential map so it can be multiplied with quaternions from EKF estimation.

$$q_{\hat{\eta}^{\mathrm{nb}}} = \exp\left(\frac{\hat{\eta}^{\mathrm{nb}}}{2}\right) \tag{3-12}$$

Let's start again from the end of the last EKF update. To be more clear, we denote the result of the standard EKF estimation at time t by $\hat{q}_{t|t}^{\mathrm{EKF}} = \hat{q}_{t|t}^{\mathrm{nb}}$, and quaternion related to RL update by $\hat{q}_{t|t}^{\mathrm{RL}}$. For simplification, the subscripts indicating the conversion relationship of the coordinate system are also omitted, all of them are the orientation between body frame b and navigation frame n. Then the second measurement update be derived as:

$$\begin{aligned} \hat{q}_{t|t}^{\mathrm{RL}} &= \exp\left(\frac{\hat{\eta}_t^{\mathrm{RL}}}{2}\right) \otimes \hat{q}_{t|t}^{\mathrm{EKF}} \\ \hat{\eta}_t^{\mathrm{RL}} &= K_t^{\mathrm{RL}} \varepsilon_t^{\mathrm{RL}} \end{aligned} \tag{3-13}$$

where the estimated measurement also updates through the newest EKF estimation.

$$\begin{gathered} \varepsilon_t^{\mathrm{RL}} = y_t - \hat{y}_{t|t}^{RL} \\ y_t = \begin{pmatrix} y_{\mathrm{a},t} \\ y_{\mathrm{m},t} \end{pmatrix}, \quad \hat{y}_{t|t}^{RL} = h\left(\hat{q}_{t|t}^{\mathrm{EKF}}\right) = \begin{pmatrix} -\hat{R}_{t|t}^{\mathrm{EKF}} g^{\mathrm{n}} \\ \hat{R}_{t|t}^{\mathrm{EKF}} m^{\mathrm{n}} \end{pmatrix} \end{gathered} \tag{3-14}$$

Here, the estimator gain $K_t^{RL} \in \mathcal{A}$ in equation (3-13) is action generated by trained RL policy, which is the component of the following MDP:

$$\hat{\eta}_t^{\mathrm{RL}} \sim \mathcal{P}\left(\hat{\eta}_t^{\mathrm{RL}} \mid \hat{\eta}_{t-1}^{\mathrm{RL}}, K_t^{RL}\right), \forall t \in \mathbb{Z}_+ \tag{3-15}$$

$\mathcal{P}\left(\hat{\eta}_t^{\mathrm{RL}} \mid \hat{\eta}_{t-1}^{\mathrm{RL}}, K_t^{RL}\right) \in \mathcal{P}$ is the state transition probability. To equip our environment with the ability to interact with agents, i.e., evaluates the value of the action-state pair, the ground truth of the trajectory is needed during the training process as the environment's inherent information. The reward function is defined as orientation deviation between the updated RL estimation and ground truth $\hat{\eta}_t^{\mathrm{RL}\to\mathrm{GT}}$. Because the larger the distance between RL estimation and ground truth, the less the goodness of the action. The distance cost is set to a negative number in the reward function, and the maximum reward will be infinitely close to 0.

$$\begin{aligned} R\left(\hat{\eta}_{t-1}^{\mathrm{RL}}, K_t^{RL}\right) &= -\mathbb{E}_{P\left(\cdot \mid \hat{\eta}_{t-1}^{\mathrm{RL}}, K_t^{RL}\right)}\left[\left\|\hat{\eta}_t^{\mathrm{RL}\to\mathrm{GT}}\right\|^2\right] \\ \hat{\eta}_t^{\mathrm{RL}\to\mathrm{GT}} &= 2\log\left(\left(\left(\hat{q}_{t|t}^{\mathrm{RL}}\right)^{\star}\right)^{R} \otimes q_{t|t}^{\mathrm{GT}}\right) \end{aligned} \tag{3-16}$$

where $(\cdot)^{\star}$ and $(\cdot)^{R}$ are conjunction and right- multiple operations for quaternion respectively.

As a summary, the complete RL estimator algorithm flow is as follows.

Algorithm 2 RL Compensated EKF (RLC-EKF) Algorithm for Orientation Estimation

INPUTS: IMU measurements $\{y_{\omega,t}, y_{a,t}\}_{t=1}^{N}$, magnetometer measurement $\{y_{m,t}\}_{t=1}^{N}$, ground truth quaternion trajectory $\{q_t^{\mathrm{GT}}\}_{t=1}^{N}$ and the initial quaternion state $q_{0|0}^{\mathrm{nb}}$
OUTPUTS: Improved orientation estimation $\hat{q}_{t|t}^{\mathrm{RL}}$.

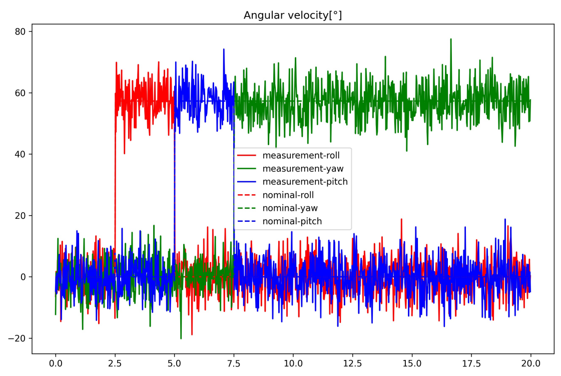
```
1: for t=1,2,3...N do
       (Prediction Update)
2:     Propagate current quaternion according to equation (3-3) with measured angular velocity.
       (EKF Correction)
3:     Reduce the uncertainly after motion update by applying the measurement update according to equation (3-5). Measurements from accelerometer and magnetometer $y_{a,t}$ and $y_{m,t}$ are needed.
       (Quaternion Re-normalisation)
4:     Normalise the quaternion and its covariance as in equation (3-11).
       (RL Correction - estimate the residual in estimation)
5:     Compute the residual $\hat{\eta}_t^{\mathrm{RL}}$ and gain $K_t^{RL}$ from equation (3-12) to (3-15), where the mapping from input $\hat{\eta}_t^{\mathrm{RL}}$ and output $K_t^{RL}$ is decided by well-trained RL policy model.
       (RL quaternion re-normalisation)
6:     Same as the post processing after EKF, re-normalize the RL result for the next iteration.
7: end for
```

3-2 Simulation

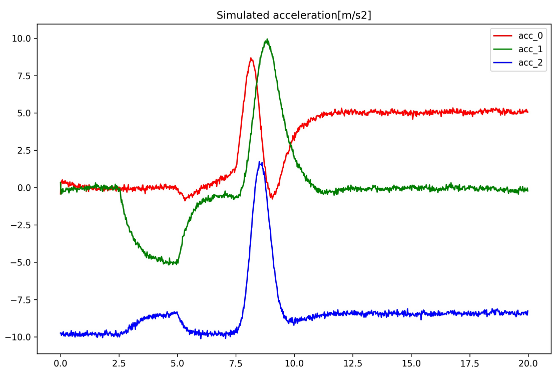
In this section, we train and evaluate our RLC-EKF framework by simulated data. First, simulated data is generated from mathematical models. Then RLC-EKF estimator is trained with selected hyperparameter settings. Finally, we inference the trained models with different orientation estimation environments. Both performance and robustness of the proposed idea are concisely compared with EKF.

3-2-1 Synthetic Data Generation

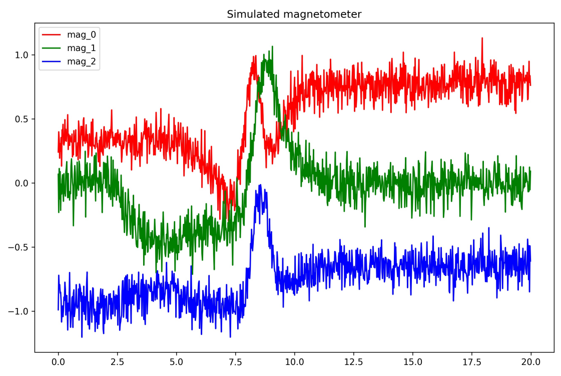
We should first prepare the synthetic data for training and evaluating our RLC-EKF estimator. Ground truth orientation is updated by recursively multiplying nominal angular velocity with respect to the true initial state.



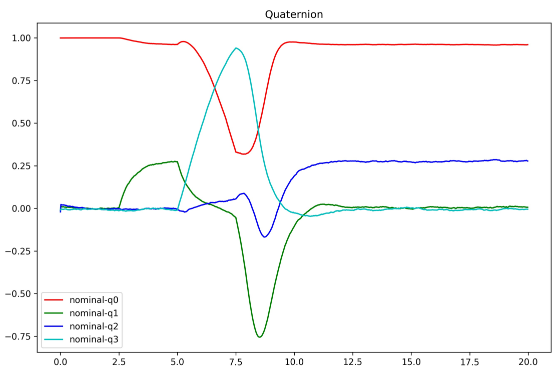
(a) Simulated angular velocity(nominal and measurement



(b) Simulated accelerometer measurement



(c) Simulated magnetometer measurement



(d) Ground truth orientation in quaternion

Figure 3-2: An example showing the composition of simulated data. With $\sigma_\omega = 1 \cdot 10^{-2}, \sigma_{\mathrm{a}} = 5 \cdot 10^{-3}, \sigma_{\mathrm{m}} = 5 \cdot 10^{-3}$. for gyroscope noise $e_{\omega,t} \sim \mathcal{N}\left(0, \sigma_\omega^2 \mathcal{I}_3\right)$, accelerometer noise $e_{\mathrm{a},t} \sim \mathcal{N}\left(0, \sigma_{\mathrm{a}}^2 \mathcal{I}_3\right)$, and magnetometer noise $e_{\mathrm{m},t} \sim \mathcal{N}\left(0, \sigma_{\mathrm{m}}^2 \mathcal{I}_3\right)$ respectively. The initial state of the quaternion trajectory is randomly selected.

IMU and magnetometer measurements are generated from sensor measurement model which is clarified in section 3-1-1. In an ideal environment, we assume accelerometer is rotated around the origin of the accelerometer triad so that accelerometer only measures gravity and magnetometer only measures magnetic field. In actual experiments, gyroscope bias can be eliminated by statistical computation. In order to make the simulation experiment as close to reality as possible, we set the bias and bias noise of the gyroscope to 0 in the simulation experiment. For the true value of gravitational acceleration and magnetic field strength, we decide $g^{\mathrm{n}} = \begin{pmatrix} 0 & 0 & g \end{pmatrix}^{\top}$ and $m^{\mathrm{n}} = \begin{pmatrix} 0.39 & 0 & -0.92 \end{pmatrix}^{\top}$ based on the 67° dip angle of the Netherlands. By setting the covariance quantities of different gyroscope noise$e^{\mathrm{b}}_{\omega,t}$, accelerometer noise$e^{\mathrm{b}}_{\mathrm{a},t}$ and magnetometer noise$e^{\mathrm{b}}_{\mathrm{m},t}$ combination, we can get different conditioned orientation estimation simulation environment.

3-2-2 Experiment Setup

Objectives

Because the simulation environment is more ideal than the reality, we have more requirements for the estimator's performance at this stage. At the same time, we also wish to explore the potential of the estimator as much as possible. We analyze the proposed framework from five perspectives and compare the results with the EKF method to see the advantages.

- **Normal estimation condition**: Whether RLC-EKF outperforms EKF with a relatively normal orientation estimation condition, i.e., medium sensor noise, model uncertainty, and reasonable initial estimate.
- **Unreliable initial estimation**: Initial estimate that is far away from truth is provided to test whether RLC-EKF is still robust when EKF may converge slower or even fail,
- **Tremendous noise disturbance**: Sensor noise intensity affects both measurements reliability and system model uncertainty, which will affect EKF's accuracy.
- **Generality**: You may question the versatility of the proposed method. For example, whether the method can be successfully trained for different angular velocity profiles or whether the policy model trained with an exact single profile still works fine when deployed on other profiles.

RL training process

The training process of the RL agent is conducted by the renowned open-source reinforcement learning platform, stable-baseline[22]. First, our RLC-EKF is wrapped as an Gym[23] standard environment, comprises of MDP components, i.e., state vector, action matrix, MLP policy, and value function. The python class of the wrapped environment has a fixed format requirement. Before the trajectory starts, the system will be reset to the initial state. Then the agent will execute the step function iteratively. By interacting with the environment, the agent shifts its state and obtains a series of feedback. According to the training settings, after a fixed number of steps being executed, the RL agent will calculate the batched reward and update the policy and value function accordingly. When the entire trajectory is completed or current estimation is significantly deviated from the truth, the agent stops and resets, which is called an episode. The training process stops once the total training step is traversed.

All RL models in this thesis is trained by PPO2 algorithm implemented in stable baselines, which enables multi vectorized environments during training comparing to PPO1. Due to the stochastic of the neural network, the training of the RL model is not guaranteed to success every time, so the training parameters also need to be adjusted in detail. The training hyperparameters used in this experiment are shown in the following table:

Table 3-1: Hyperparameters of PPO2 training process (orientation estimation)

Hyperparameters	Value
Episodes	500
Time horizon	1000
Optimization batch size	128
Learning rate	$2.5e^{-4}$
Value function coefficient	$3e^{-4}$
Discount γ	0.99
Clip range ϵ	0.2
MLP dimension of π_ϕ	$(64, 64)$
MLP dimension of V_θ	$(64, 64)$
State space range	$[-10, 10]$
Action space range	$[-0.02, 0.02]$

The shape of the state vector and action vector is $(3,)$ and $(18,)$ respectively, corresponding to the shape of our error state $\hat{\eta}_t^{\mathrm{RL}}$ and estimator gain K_t^{RL}. The action vector is reshaped to $(3X6)$ for matrix multiplication. During our experiments, the range of the RL action space has an essential impact on the success rate of model training and model performance. That makes sense since our action space is continuous. When the range is too large, it is difficult for the agent to narrow down to the correct action quickly. While a too-small range will weaken RL's compensation for residual errors, thereby reducing performance. We calculate the success rate of training by the scale of available models among 20 trained models.

Table 3-2: Selection of action space

Action range	$[-0.01, 0.01]$	$[-0.02, 0.02]$	$[-0.03, 0.03]$	$[-0.05, 0.05]$	$[-0.1, 0.1]$
Training success rate	0.75	**0.8**	0.6	0.25	0.05

In the following sections, we train each scenario with ten policies and select the RL policy with the lowest validation error. This best-performed model is then deployed for inferencing unknown environments. Based on different experimental purposes, more specific experimental settings will be introduced in correspondent subsections.

3-2-3 Normal Estimation Condition

For this scenario the model is trained with medium noise level where $\sigma_\omega = 1 \cdot 10^{-2}, \sigma_{\mathrm{a}} = 1 \cdot 10^{-3}, \sigma_{\mathrm{m}} = 1 \cdot 10^{-3}$ for gyroscope noise $e_{\omega,t} \sim \mathcal{N}\left(0, \sigma_\omega^2 \mathcal{I}_3\right)$, accelerometer noise $e_{\mathrm{a},t} \sim \mathcal{N}\left(0, \sigma_{\mathrm{a}}^2 \mathcal{I}_3\right)$, and magnetometer noise $e_{\mathrm{m},t} \sim \mathcal{N}\left(0, \sigma_{\mathrm{m}}^2 \mathcal{I}_3\right)$ respectively. The sampling rate is 100HZ and the whole trajectory lasts for 10 seconds. In the training process, at the beginning of each trajectory, true quaternion state q_0^{nb} is randomly sampled from uniform distribution $\mathbb{U}([-1,-1,-1,-1],[1,1,1,1])$, and initial estimation $\hat{q}_0^{\mathrm{nb}}$ is added with an extra uniform distribution $\mathbb{U}([-0.5,-0.5,-0.5,-0.5],[0.5,0.5,0.5,0.5])$ on the basis of q_0^{nb}. By traversing different quaternion states, RL learns how to eliminate errors through secondary updates when EKF converges. We use a non-trivial one direction rotation profile which is shown as follows:

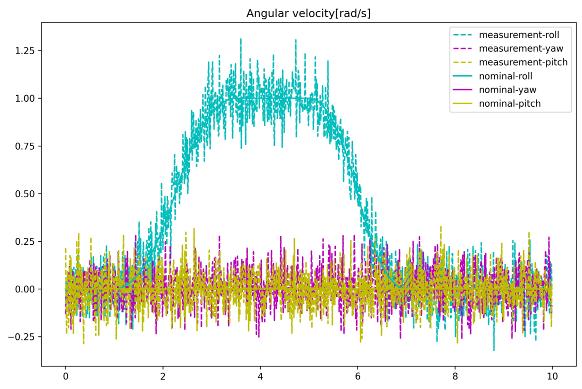


Figure 3-3: Angular velocity profile used for training

The quantitative performance of the trained model is shown as follows. The results are based on 100 Monte Carlo simulation experiments. Each trial starts with a random initial estimation. The results are expressed in RMSE(root-mean-square error) of independent Euler angles and quaternion error we defined in equation (3-16).

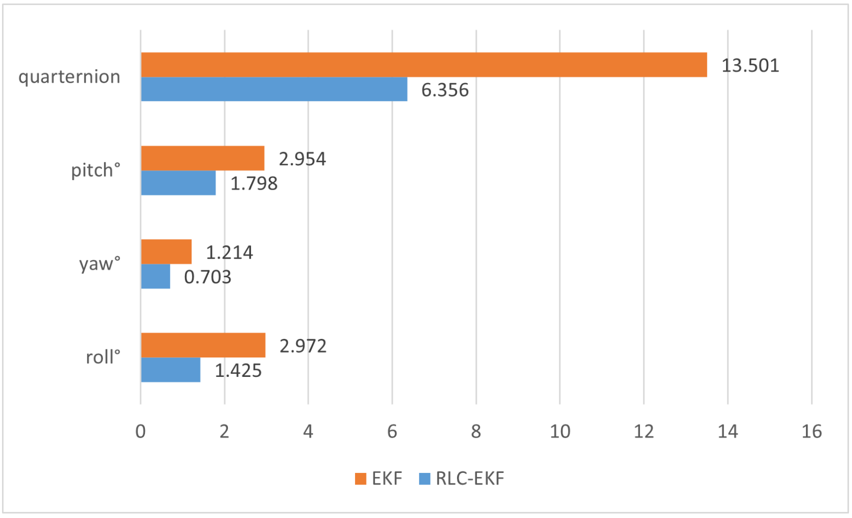
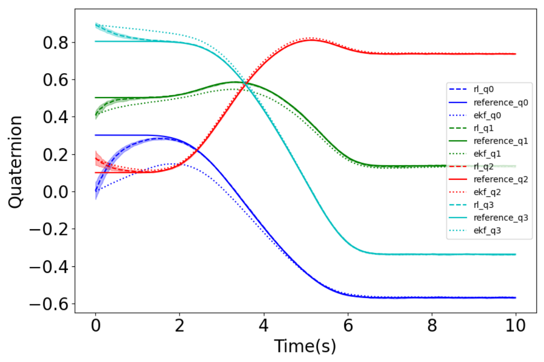
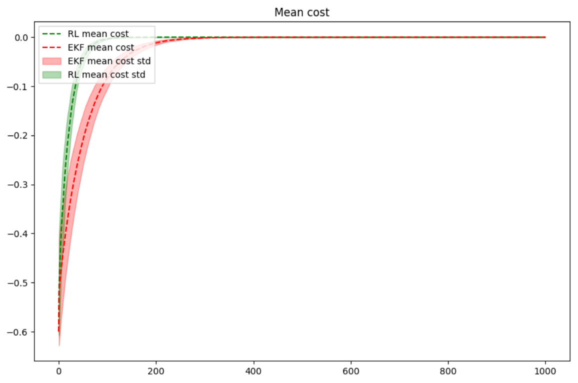


Figure 3-4: RMSE of the orientation estimation in normal condition

To visualize the orientation tracking process and RL reward (equivalent to estimation cost), we choose a set of fixed initial true estimate, evaluate the trajectory ten times to eliminate randomness. From the results, we can say RLC-EKF successfully compensates for the residual error in EKF estimation, especially at the beginning estimation stage. Thus leads to faster convergence and higher overall accuracy.



(a) Orientation tracking

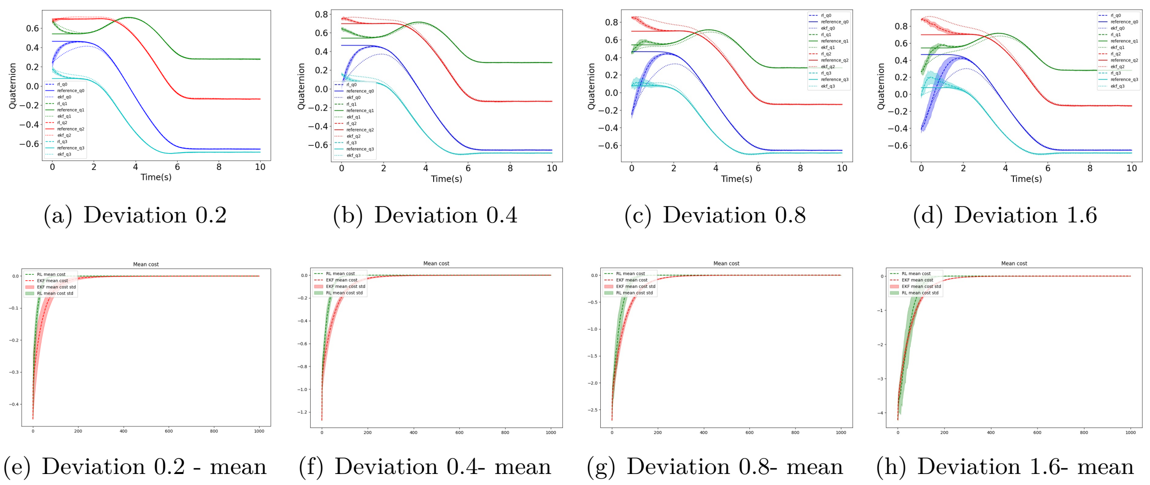


(b) Quaternion error

Figure 3-5: Comparison of RLC-EKF and EKF tracking accuracy. 4 elements in unit quaternion of reference truth(solid line), EKF estimation(dotted line) and RLC-EKF(dashed line) are displayed. The trajectories are the mean path value among 10 trials, with shaded area representing standard deviation. The cost is in negative order.

3-2-4 Unreliable Initial Estimation

In this scenario, we use the trained model from the last section but evaluate it on other randomly sampled initial estimates. By traversing different initial states, RL policy learns how to quickly keep up with the trajectory in the initial stage. To evaluate whether the trained model is robust with unreliable initialization, the deviation between the initial estimate and true state grows exponentially according to the ratio of 1:2:4:8. The smallest deviation is [0.2, -0.2, 0.2, -0.2], before unit normalization. The largest deviation range is not even included in the training process, which will testify the availability of the RL agent under the unexplored environment.



(a) Deviation 0.2 (b) Deviation 0.4 (c) Deviation 0.8 (d) Deviation 1.6

(e) Deviation 0.2 - mean (f) Deviation 0.4- mean (g) Deviation 0.8- mean (h) Deviation 1.6- mean

Figure 3-6: Orientation with growing initial estimate deviation. Line type description is the same manner as section 3-2-3.

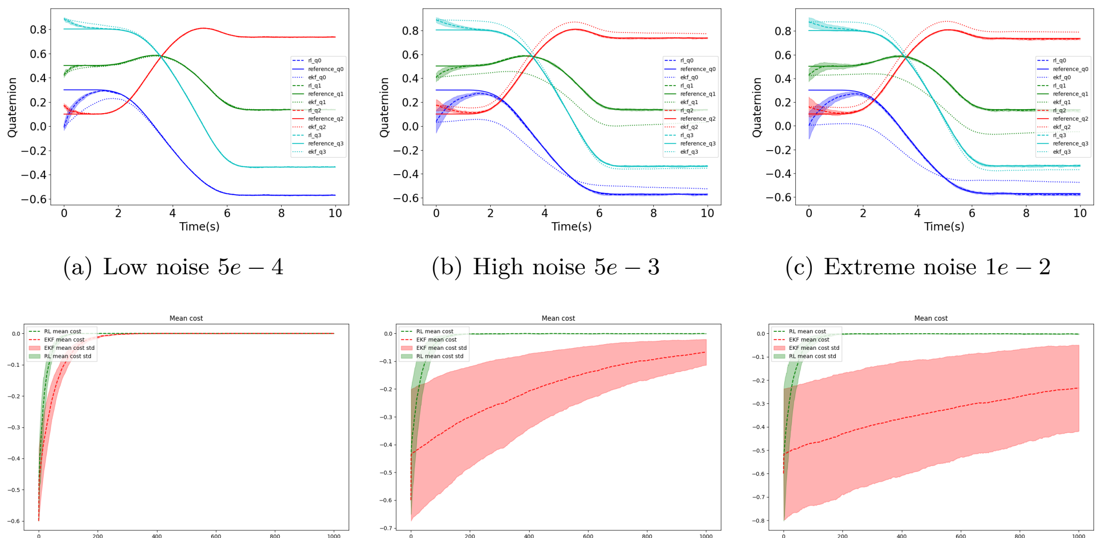
From results we can tell that when initial state deviation increases, The uncertainty of the RLC-EKF estimation also increases, in rare cases it performs worse than EKF. But the overall mean accuracy of RLC-EKF still outperforms EKF, proving its capability of handling unreliable initial estimation.

Table 3-3: Quaternion RMSE of RLC-EKF w.r.t initial deviation

Deviation	0.2	0.4	0.8	1.6
RLC-EKF	5.867	21.34	73.23	179.94
EKF	16.45	51.48	132.71	221.19
Advantage ratio	2.80	2.41	1.81	1.22

3-2-5 Tremendous Noise Disturbance

In this scenario, the trained model stays the same and is deployed in four different measurement noise environments. Measurement noise also comprises model uncertainty, which will defect the performance of EKF significantly. We keep the accelerometer and magnetometer noise consistent and gradually increase their magnitude in the order of $[5e-4, 1e-3, 5e-3, 1e-2]$.



(a) Low noise $5e-4$ (b) High noise $5e-3$ (c) Extreme noise $1e-2$

(d) Low noise $5e-4$ - mean (e) High noise $5e-3$ - mean (f) Extreme noise $1e-2$- mean

Figure 3-7: Orientation tracking under chaotic noise environment. The result for medium noise $1e-3$ is already given in figure 3-6. Line type description is the same manner as section 3-2-3.

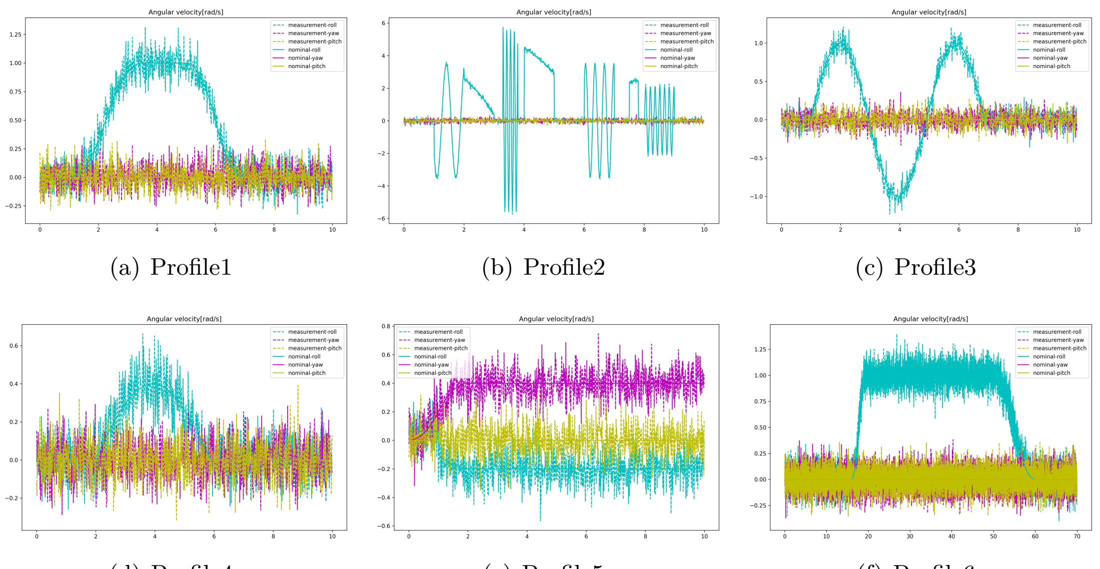
From results, we can observe the performance of EKF is severely damaged by enormous measurement noise. EKF doesn't even converge in high and extreme noise conditions. Meanwhile, even under the chaotic environment, RLC-EKF still gives an acceptable performance. RL obtains more benefits in large noise conditions as there is more residual error from EKF to be compensated.

Table 3-4: Advantage of RLC-EKF w.r.t different measurement nose level

Noise level	$Low(5e-04)$	$Medium(1e-03)$	$High(5e-03)$	$Extreme(1e-02)$
RLC-EKF	7.08	8.89	19.35	17.728
EKF	15.1	32.32	213.23	327.51
Advantage ratio	2.13	3.63	11.01	18.47

3-2-6 Generality

In this scenario, we discuss the applicability of the proposed method in different tracking environments, i.e., angular velocity profiles. Since we use a trained model from the same angular velocity profile in the previous scenarios, the RL agent may only perform well in specific paths but fails when profiles change. This situation shouldn't happen theoretically because we design our RL update according to the error Euler angle state. RL is only sensitive to residual errors, so there should be no obvious difference between different profiles. To validate this, the well-trained model from profile1 in Figure 3-4 is executed on five other angular velocity profiles shown below.

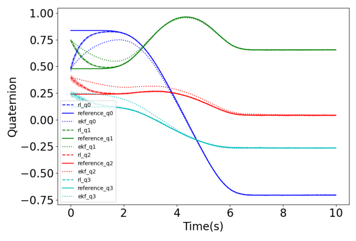


(a) Profile1 (b) Profile2 (c) Profile3

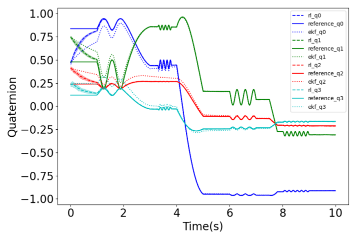
(d) Profile4 (e) Profile5 (f) Profile6

Figure 3-8: Different angular velocity profiles used in this scenario.

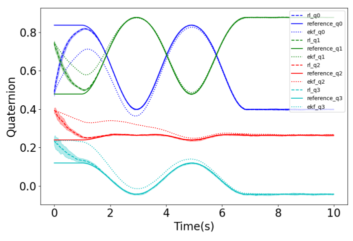
Notice that the last profile lasts for 70s instead of 10s to evaluate the performance under long tracking period. We choose medium level noise $1e-3$, initial deviation [0.5, -0.5, 0.5, -0.5] for every trajectory, the results are shown as follows.



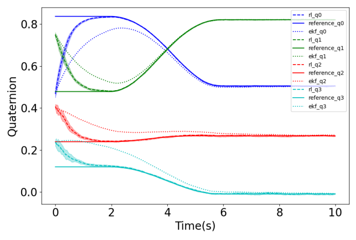
(a) Tracking in profile1



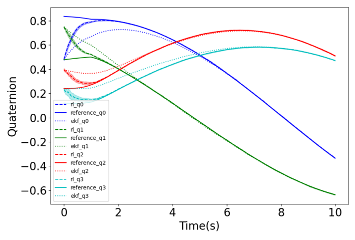
(b) Tracking in profile2



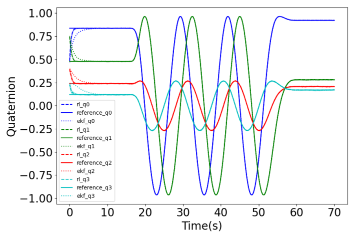
(c) Tracking in profile3



(d) Tracking in profile4



(e) Tracking in profile5



(f) Tracking in profile6

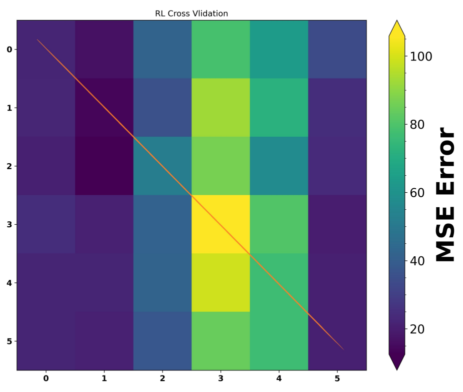
Figure 3-9: Same profile trained from profile1 evaluating on other profiles.

Table 3-5: RLC-EKF evaluating on different profiles

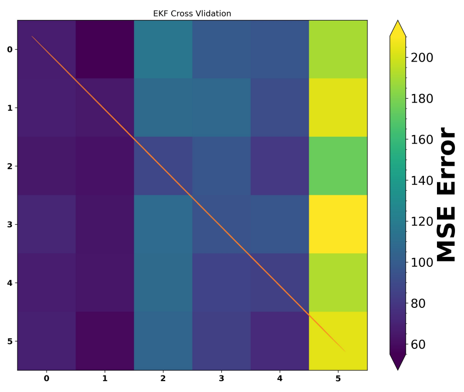
Angular velocity	Profile1	Profile2	Profile3	Profile4	Profile5	Profile6
RLC-EKF	22.365	22.446	20.575	24.85	22.114	22.109
EKF	66.657	68.289	64.72	71.742	67.159	68.532

Cross Validation

Above result shows the generality of the model trained from exact one profile on other profiles. In order to eliminate the coincidence that the RL model is only trainable on the selected profile1, profile 2-5 listed in Figure 3-8 are also trained and cross-validated with each other. The quantitative results is shown in a matrix visualization graph below:



(a) RLC-EKF



(b) EKF

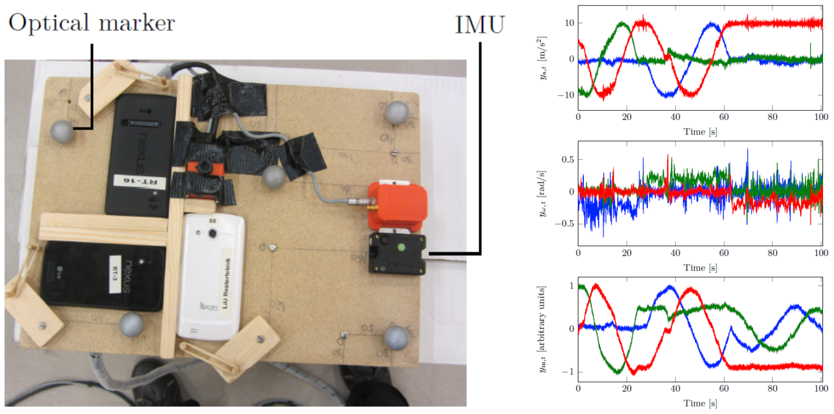
Figure 3-10: Result visualization for profiles benchmark

By comparing the diagonal and off-diagonal data, we can observe that the performance of the model on the training profile is not necessarily the best. The error of the same model on different profiles is also within a reasonable range. Based on this, we can finally conclude the proposed algorithm has excellent generality for orientation estimation tasks.

3-3 Real Dataset

After concisely evaluating the RLC-EKF in experimental conditions, it's time to test the validity in practice. A dataset containing ground truth orientation, raw IMU, and magnetometer measurements is used for training and evaluating the RL agent. The ground truth orientation is provided through optical markers which are tracked by multiple cameras. While measurement data is collected from the Trivisio Colibri Wireless IMU with a logging rate of 100Hz. The dataset is already well synchronized and ready for experiments when we got it. We thank Manon Kok, who collects and provides the data.

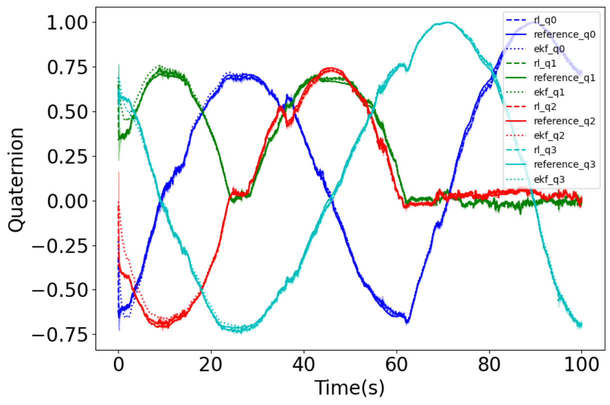
We train the model through the first half of the dataset. And evaluate on the rest of the dataset. The results show that RLC-EKF still performs well in the real environment.



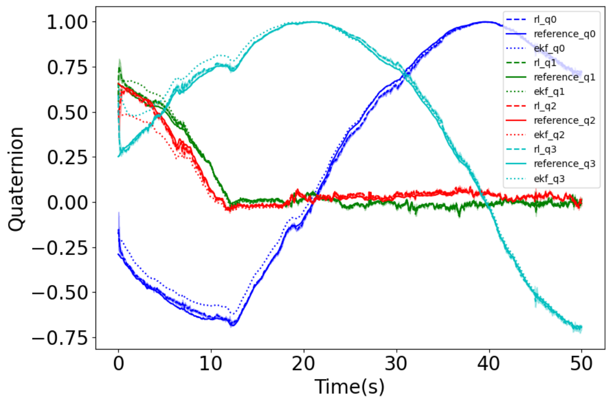
(a) Hardware experiment setting (b) Sensor measurements

Figure 3-11: Real dataset collection process.

We evaluate the best trained model on whole dataset (last 100s) and last half dataset (last 50s), respectively, The result is shown below. Note that because the measurement data is fixed noise so the EKF estimation is deterministic in Monte Carlo Simulation. That's why there is no standard deviation filling for EKF curves in qualitative results.



(a) Whole dataset(0-100s)



(b) 50-100s

Figure 3-12: Real data quaternion tracking result.

Table 3-6: RLC-EKF on real dataset

RMSE (Total)	Yaw[°]	Pitch [°]	Roll[°]	Quaternion	Yaw[°]	Pitch [°]	Roll[°]	Quaternion
	Whole dataset				50s-100s			
RL	3.425	1.334	6.267	49.738	3.696	1.101	2.583	28.666
EKF	3.992	1.54	4.534	125.343	9.237	1.487	13.342	163.907

Chapter 4

RL Estimator for Localization

In this chapter, instead of estimating orientation solely, we will implement the proposed RLC-EKF idea on a 2D ground robot localization scenario. We focus our scenario on a feature-based localization task within the static known map. The chapter organization is similar to chapter 3. First, the integrated RL framework based on a classical EKF algorithm is introduced. Then the framework is trained and evaluated on both simulated and real data. The results are compared with that of EKF solely in terms of both accuracy and robustness. The standard feature-based localization EKF algorithm is derived from the textbook Probabilistic Robotics[24], Chapter 7.

4-1 RLC-EKF for Localization

Localization is the most basic perceptual manipulation for an intelligent ground robot. The most fundamental SOI of a plane localization task is $x_t = \begin{pmatrix} x & y & \theta \end{pmatrix}^T$. If we assume the uniform in z-direction, the absolute x, y position plus the heading angle(yaw) are sufficient to locate the robot within a known scale space, namely map. Sometimes velocity and acceleration are also included in the state vector depending on the task requirements. A map can be either static or dynamic. In a dynamic map, surrounding objects such as pedestrians or cars are moving all the time. To utilize them for localizing will also require our robot to add the coordinates of these objects to SOI for tracking them continuously, which seriously degrades the accuracy and efficiency of the algorithm. The input of EKF's measurement update can be either raw sensor measurement or pre-processed features. Features are usually the specific geometric pattern of the object, such as lines, corners, centroids. Feature measurements take advantage of less computation while keeping the most valuable information about surroundings. Overall, as this is the first time we implement our RLC-EKF filter on a localization task, we limit our scene to be *static map* and *feature measurements.* The success of this experiment will provide guidance for subsequent more complex scenarios.

4-1-1 Feature-based EKF Localization

In this section, we introduce a standard feature-based EKF localization algorithm. The premise of the algorithm is a static map with known correspondence, i.e., all surrounding landmarks' true coordinates are known in advance. Meanwhile, each measurement contains a signature for the correspondence to the landmark it belongs to. With the known correspondence assumption, there is no need to match the measurements with the landmarks, namely data association, which will bring enormous error and computation effort. This assumption makes sense in the real application if there is landmark identity detection when acquiring the measurements. Within the known map, the robot first propagates its state through a self-motion model. Then based on the innovations between the observed and actual measurements of several landmarks, EKF corrects the robot's belief of its state and gradually narrows down to the actual true state. Note that such a general framework applies to various sensors, e.g., cameras, laser range finders, ultra-wideband (UWB), as long as they can provide feature measurements with range and bearing information.

We start with the standard EKF belief $x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = \begin{pmatrix} \mathbf{x}_0 & \mathbf{y}_0 & \theta_0 \end{pmatrix}^T$ and $\Sigma_0 = I_{3x3}$ are the mean and covariance of the state's multivariate Gaussian distribution respectively.

Motion update

The odometry motion model $g(\cdot)$ of the robot is given as follows:

$$\begin{gathered}\overline{\mu_t} = g\left(u_t, x_{t-1}\right) = \mu_{t-1} + \begin{bmatrix} v_t \Delta t \cos\left(\theta + \frac{\omega_t \Delta t}{2}\right) \\ v_t \Delta t \sin\left(\theta + \frac{\omega_l \Delta t}{2}\right) \\ \omega_t \Delta t \end{bmatrix} \\ \overline{\Sigma_t} = G_t \Sigma_{t-1} G_t^T + R_t \end{gathered} \tag{4-1}$$

Where $u_t = \begin{bmatrix} v_t & \omega_t \end{bmatrix}^T$ is the actual odometry input includes transnational velocity v_t and rotational angular velocity ω_t. Note that this actual input already contains Gaussian noise $e_{u,t} \sim \mathcal{N}(0, R_t)$, compared to nominal odometry input. R_t is the covariance matrix of the input noise. As long as the time interval Δt is small enough, the difference between the distance of the sequential frame can be approximated by trigonometric projection, which is illustrated in figure 4-1(a).

Here, G_t is the linearized function of $g(\cdot)$, which is given by:

$$G_t = \frac{\partial g\left(u_t, \mu_{t-1}\right)}{\partial x_{t-1}} = \begin{bmatrix} 1 & 0 & -v_t \Delta t \sin\left(\theta + \frac{\omega_t \Delta t}{2}\right) \\ 0 & 1 & v_t \Delta t \cos\left(\theta + \frac{\omega_t \Delta t}{2}\right) \\ 0 & 0 & 1 \end{bmatrix} \tag{4-2}$$

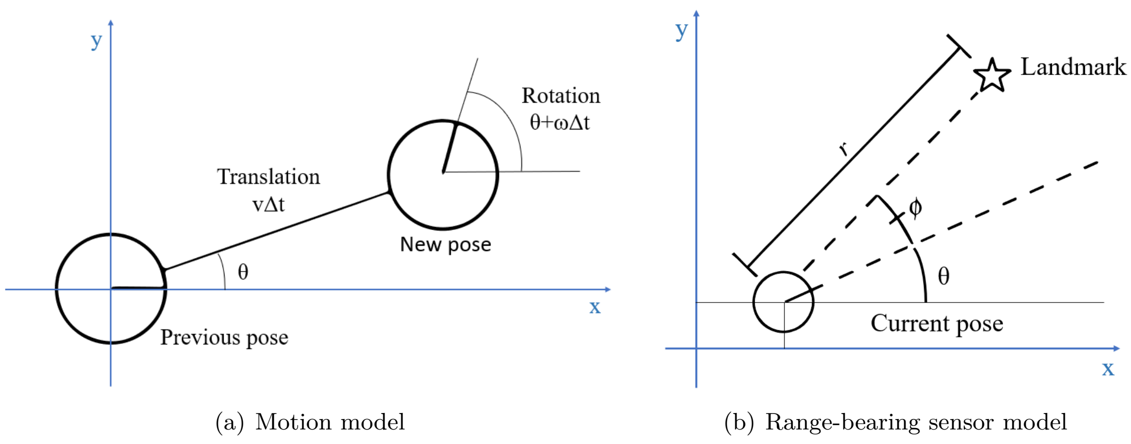


Figure 4-1: Illustration of dynamic and measurement models for ground localization task

Measurement update

The state uncertainty Σ_t always expands after odometry motion update, that's why we utilize feature measurements to reduce the uncertainty. In this framework we regard the sensor measurements are in a range-bearing form, which is illustrated in figure 4-1(b). A measurement attached to the specific landmark j at time t can be expressed by $z_{j,t} = \begin{pmatrix} r_{j,t} & \phi_{j,t} & c_j \end{pmatrix}^T$. The range r (straightforward distance between robot and landmark) is calculated in Cartesian coordinate system. While the bearing ϕ (relative rotation) is calculated by arctangent function. c_j is the correspondence matching this measurement to the landmark j. The measurement function $f(\cdot)$ is then given by:

$$\begin{gathered} \hat{z}_t^i = h(x_t, m) = \begin{bmatrix} \sqrt{q} \\ \operatorname{atan2}(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \\ m_{j,s} \end{bmatrix} \\ q = (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2 \end{gathered} \tag{4-3}$$

Where $m_{j,x}, m_{j,y}$, and $m_{j,s}$ are the ground truth x position, y position, and identity of the landmark j, respectively. Through the measurement model, we can convert current estimation $\bar{\mu}_t$ into the *should-have-seen* range-bearing measurement form. This measurement can then be compared with the actual observed measurement. The EKF update process for a single measurement is given as:

$$\begin{aligned} \mu_t &= \bar{\mu}_t + K_t^j \left(z_t^j - \hat{z}_t^j \right) \\ \Sigma_t &= \left(I - K_t^j H_t^j \right) \bar{\Sigma}_t \\ S_t^j &= H_t^j \bar{\Sigma}_t \left[H_t^j \right]^T + Q_t \\ K_t^j &= \bar{\Sigma}_t \left[H_t^j \right]^T \left[S_t^j \right]^{-1} \end{aligned} \tag{4-4}$$

Where z_t^j is the real sensor measurement at time t. Q_t is the noise covariance matrix of the sensor. H_t^j is the differential of the non-linear function $h(\cdot)$ given in:

$$H_t^i = \frac{\partial h(\bar{\mu}_t, m)}{\partial x_t} = \frac{1}{q}\begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 \\ \delta_y & \delta_x & -1 \\ 0 & 0 & 0 \end{pmatrix}$$
$$\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} m_{j,x} - \bar{\mu}_{t,x} \\ m_{j,y} - \bar{\mu}_{t,y} \end{pmatrix} \tag{4-5}$$

4-1-2 RLC-EKF Framework

In most cases, there is not only one measurement value but several values for multiple landmarks. We can either update them one by one recursively or stack them together and update them at once. Considering the RL agent only interacts with the environment once at each timestamp, we stack up all available measurements together and update them as a whole. However, the number of measurements is dynamic at each timestamp, different from the promising single measurement we deal with in the orientation estimation experiment. Suppose we want the RL agent to learn the Kalman gain matrix as the action. In that case, the size of the action space should be fixed during the whole training and evaluation process, which restricts us to only update the estimation with a fixed number of measurements at each step.

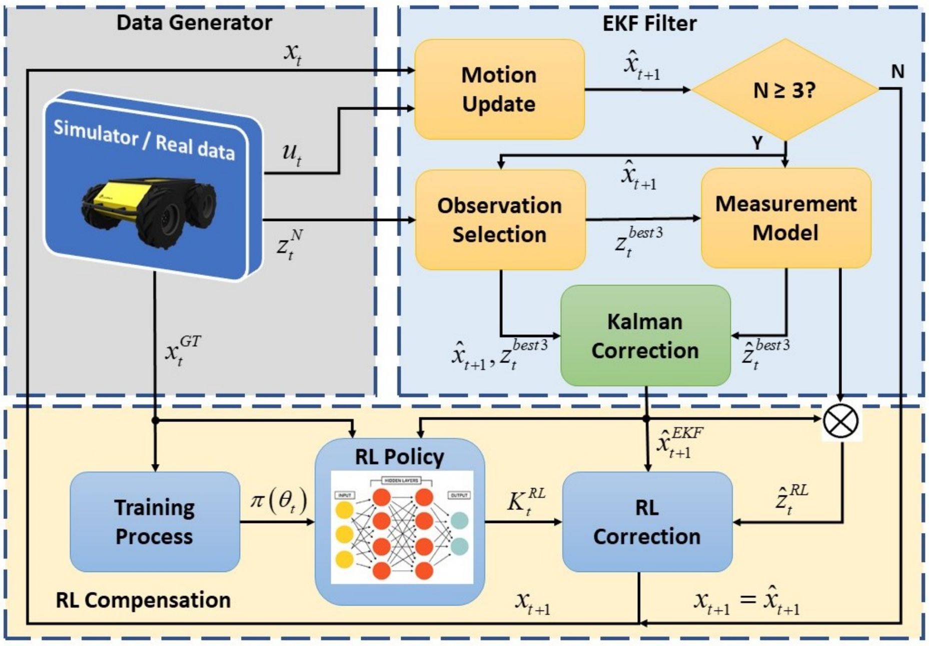


Figure 4-2: RLC-EKF structure for Localization

To solve the issue, We have developed a strategy for judging whether the RL agent is activated for compensating. When the number of current timestamp measurements is greater than or equal to **three**, the RL agent selects the most suitable three measurement values for the compensation update of EKF. When the measured value is less than three, RL stops updating, and the estimation of EKF serves as the beginning state of the next cycle. The selection of the number three is inspired by [25], where the trade-off between the computation cost and performance of the EKF is discussed in terms of the measurement instance numbers. When measurement numbers exceed three, the selection criteria are conducted. This criterion is inspired by part of the K-nearest neighbor (KNN) [26] algorithm. Instead of selecting the closest measurements, we select the three furthest measurements away from the current estimated state and utilize them for updating. The stacked update function is summarized as follows, especially for the 3-measurements batch. Superscripts indicate the shape of the matrix. Since the differential of the sensor model in terms of angle θ is always zero, the k matrix of each measurement is 6X2, independent of each other.

$$\begin{aligned}
\mu_t^{3X1} &= \bar{\mu}_t^{3X1} + K_t^{3X6}\left(z_t^{6X1} - \hat{z}_t^{6X1}\right) \\
\bar{\Sigma}_t^{3X3} &= \left(I^{3X3} - K_t^{3X6} H_t^{6X3}\right)\bar{\Sigma}_t \\
S_t^{6x6} &= H_t^{6X3}\bar{\Sigma}_t^{3X3}\left[H_t^{6X3}\right]^T + Q_t^{6X6} \\
K_t^{3X6} &= \bar{\Sigma}_t^{3X3}\left[H_t^{6X3}\right]^T\left[S_t^{6X6}\right]^{-1}
\end{aligned} \tag{4-6}$$

Now get back to the RL agent design. Similar to section 3-1-3, to be more clear, the result of from the lastest EKF estimation and RL-compensated estimation are denoted as $\bar{\mu}_t^{\mathrm{EKF}}$ and $\bar{\mu}_t^{\mathrm{RL}}$ respectively. And RL update is expressed by:

$$\begin{aligned}
\mu_t^{\mathrm{RL}} &= \bar{\mu}_t^{\mathrm{EKF}} + \varepsilon_t^{\mathrm{RL}} \\
\varepsilon_t^{\mathrm{RL}} &= K_t^{\mathrm{RL}}(z_t - \hat{z}_t^{\mathrm{RL}}) \\
\hat{z}_t^{\mathrm{RL}} &= h\left(\bar{\mu}_t^{\mathrm{RL}}\right)
\end{aligned} \tag{4-7}$$

A markov decision process can also be formulated:

$$\varepsilon_t^{\mathrm{RL}} \sim \mathcal{P}\left(\varepsilon_t^{\mathrm{RL}} \mid \varepsilon_{t-1}^{\mathrm{RL}}, K_t^{\mathrm{RL}}\right), \forall t \in \mathbb{Z}_+ \tag{4-8}$$

Here $K_t^{RL} \in \mathcal{A}$ is a 3X6 action matrix served as the estimator gain. Also the reward function is defined as:

$$\begin{aligned}
R\left(\varepsilon_{t-1}^{\mathrm{RL}}, K_t^{RL}\right) &= -\mathbb{E}_{P\left(\cdot\mid\varepsilon_{t-1}^{\mathrm{RL}}, K_t^{RL}\right)}\left[\left\|\varepsilon_t^{\mathrm{RL}\to\mathrm{GT}}\right\|^2\right] \\
\varepsilon_t^{\mathrm{RL}\to\mathrm{GT}} &= \sqrt{\left(x_t^{\mathrm{GT}} - \bar{\mu}_{t,x}^{\mathrm{RL}}\right)^2 + \left(y_t^{\mathrm{GT}} - \bar{\mu}_{t,y}^{\mathrm{RL}}\right)^2}
\end{aligned} \tag{4-9}$$

Note that the difference in angle is not included in reward function since due to the map scale the magnitude orders of angle and translation can be huge different. At this point, we have established a complete RLC-EKF localization framework, the whole algorithm can be summarized as follows:

Algorithm 3 RL-Compensated EKF (RLC-EKF) Algorithm for Planar Localization

INPUTS: Odometry input $\{u_t\}_{t=1}^{N}$, Range-bearing measurement $z_t = \left[\begin{array}{cccc} z_t^1 & z_t^2 & \dots & z_t^k \end{array}\right]$ where k is the number of measurements at time t, Ground truth path $\{x_t^{\mathrm{GT}}, y_t^{\mathrm{GT}}\}_{t=1}^{N}$, Known map $m = \left[\begin{array}{cccc} m_1 & m_2 & \dots & m_n \end{array}\right]$ where n is the number of landmarks, Initial estimate $x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$
OUTPUTS: Compensated pose estimation μ_t^{RL}.

```
1:  for t=1,2,3...N do
        (Motion update)
2:      Propagate $\mu_{t-1}^{\mathrm{EKF}}$ and $\Sigma_{t-1}$ through motion dynamics as in (4-1).
3:      if Number of measurements > 3, then
            (Selection Criteria)
4:          for all observed features $z_t^i = \left[\begin{array}{ccc} r_t^i & \phi_t^i & s_t^i \end{array}\right]^T$ do
5:              Store $d = \sqrt{\left(z_{t,x}^j - \bar{\mu}_{t,x}\right)^2 + \left(z_{t,y}^j - \bar{\mu}_{t,y}\right)^2}$
            into a list and sort it with the high to low order.
6:          end for
            (EKF Correction)
7:          Pick out the furthest three measurements.
8:          Update $\bar{\mu}_t$ and $\bar{\sigma}_t$ by (4-4).
            (RL compensation)
9:          Based on last step error state $\varepsilon_{t-1}^{\mathrm{RL}}$, generate the estimator gain matrix $K_t^{\mathrm{RL}}$ through
    the trained policy.
10:         Calculate current error state $\varepsilon_t^{\mathrm{RL}}$ and compensate the residual to the EKF estima-
    tion $\mu_t^{\mathrm{RL}} = \bar{\mu}_t^{\mathrm{EKF}} + \varepsilon_t^{\mathrm{RL}}$.
11:     else
            (Skip update)
        Directly utilize the estimation from the motion update for the next iteration.
        $\mu_t^{\mathrm{RL}} = \bar{\mu}_t^{\mathrm{EKF}}$
12:     end if
13: end for
```

4-2 Simulation

In this section, we validate the proposed algorithm in a simulated environment. Synthetic data is generated with Gaussian noise distribution. The policy model is then trained and evaluated based on different test criteria.

4-2-1 Synthetic Data Generation

Experiment field

First, we plan a 25m by 25m square plain area as our test site. In the field, the robot starts from the initial state $\mu_0 = \left(\begin{array}{ccc} 0 & 0 & 0 \end{array}\right)^T$ and execute a circular motion with a radius of 10m.

There are a total of 20 landmarks around the circumference, with fixed, known positions. In orientation estimation experiment, we train the RL policy model with several different angular velocity profiles. We have made improvements on this basis. In order to drive the robot with different random trajectories for each training episode, we generate the landmark set alongside the robot trajectory randomly per episode. This "semi-dynamic" map setting corresponds to the scenario where the robot drives through different trajectories in a fixed map. The coordinate range of the randomly generated landmark is limited to a circle with an offset of [-2.5m, 2.5m] from the circular trajectory of the robot. In this experiment, we also assume the ideal situation that at each step, the robot has at least three sensor measurements by setting the sensor observation range as 10m, which guarantees the activation of the RL update process.

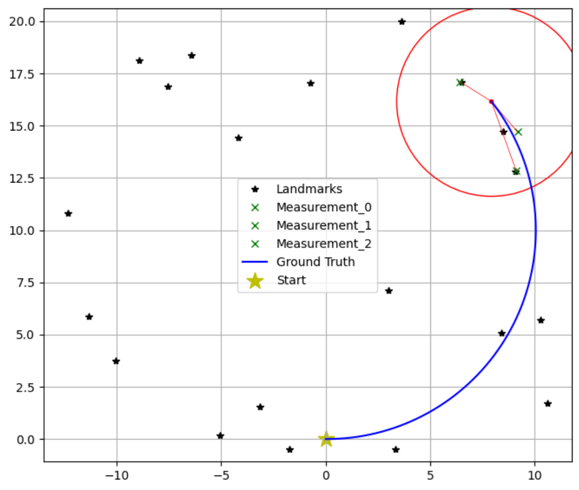


Figure 4-3: Visualization of simulation environment

Odometry input

Odometry data includes forward velocity and angular velocity of the robot at the time, given by:

$$u_t = \bar{u}_t + e_{u,t} = \begin{pmatrix} \bar{v}_t + v, t \\ \bar{\omega}_t +_{\omega,t} \end{pmatrix} \tag{4-10}$$

where $\bar{v}_t = 1m/s$, $\bar{\omega}_t = 0.1rad/s$ are the nominal values for ground truth circular path. $e_{u,t} \sim \mathcal{N}(0, R_m)$, $R_m = \begin{bmatrix} 1 & 0 \\ 0 & 0.1745 \end{bmatrix}$ adds the Gaussian noise with the range of $1m/s$ and $10°$ on $\bar{v}_t$ and $\bar{\omega}_t$ respectively.

Sensor measurement

Sensor measurement includes relative range and bearing between the sensor and landmark. At each step, simulated observations of each landmark is generated based on measurement model (4-3), given by:

$$z_t^i = \begin{bmatrix} \sqrt{\left(m_{j,x} - x_t^{GT}\right)^2 + \left(m_{j,y} - y_t^{GT}\right)^2} \\ \operatorname{atan2}\left(m_{j,y} - y_t^{GT}, m_{j,x} - x_t^{GT}\right) - \bar{\mu}_{t,\theta} \end{bmatrix} + e_{z,t} \tag{4-11}$$

where $e_{z,t} \sim \mathcal{N}(0, Q_m)$, $Q_m = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.01745 \end{bmatrix}$ adds the Gaussian noise of $1m/s$ and $1°$ on range and bearing measurement respectively. Note that this measurement is already matched with a landmark so no data association process is needed.

4-2-2 Experiment Setup

RL training process

We continue to use PPO2 algorithm from open source reinforcement learning platform *stable-baseline* to train and evaluate our RL model. The sample rate of the synthetic data is set as 10 Hz. Each trajectory lasts for 50s so 500 steps for a complete RL training episode. For each experiment setting, we train 10 models, each for **300** episodes, and pick out the best model which has based on evaluation performance. All results shown in this chapter is obtained by inferring the best model with **100** times Monte Carlo simulation. Since we update three measurements per step, the shape of the action space is the same as the orientation estimation task, i.e. 3X6 matrix. We change the setting of the network layer dimension, range of the action and observation space. After several attempts, the most appropriate training set up for plane localization task is summarized as follows:

Table 4-1: Hyperparameters of PPO2 training process (localization task)

Hyperparameters	Value
Time horizon	500
Optimization batch size	500
Learning rate	2.5e^{-4}
Value function coefficient	3e^{-4}
Discount γ	0.98
Clip range ϵ	0.2
MLP dimension of π_ϕ	$(256, 128, 64)$
MLP dimension of V_θ	$(256, 128, 64)$
State space range	$[-25, 25]$
Action space range	$[-0.002, 0.002]$

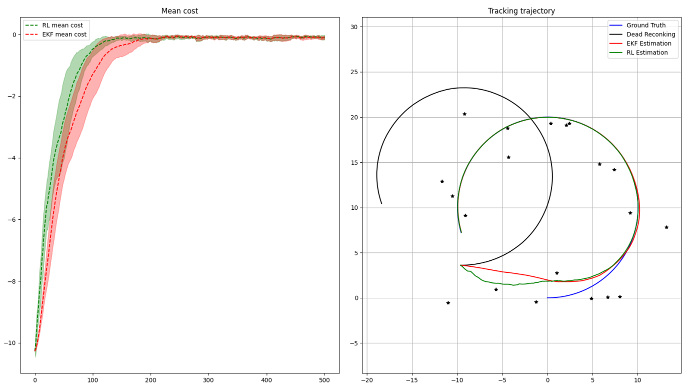
Similar to the orientation estimation task, we set several criteria to evaluate the availability of the proposed algorithm, with the consideration of **initial estimate deviation**, **measurement noise level** and **EKF model calibration**. Details are shown in the sequential sections.

4-2-3 Normal Condition

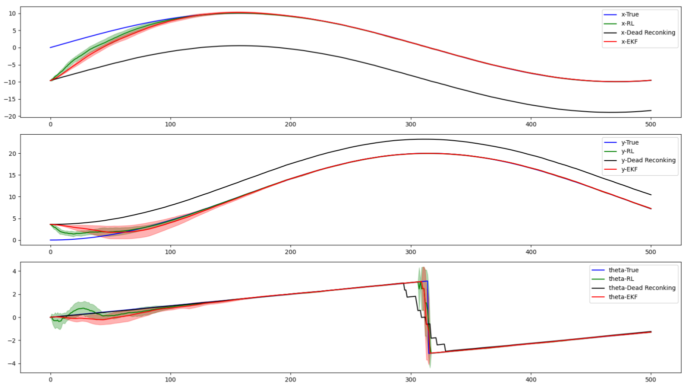
We first train and test the RL policy under the less difficult situation. Based on our evaluation criteria, the variables affecting the task difficulty are noise intensity, model covariance and initial estimation. For the moderate setting, we set initial estimate within the square area ranging from [-5m, 5m] around the original site, and input noise covariance R_m, measurement noise covariance Q_m, process model covariance R_t and measurement model noise Q_t as:

$$
\begin{aligned}
R_m &= \begin{bmatrix} 1 & 0 \\ 0 & 0.1745(10^\circ) \end{bmatrix}, \quad Q_m = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.01745(1^\circ) \end{bmatrix} \\
R_t &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.523(30^\circ) \end{bmatrix}, \quad Q_{t,j} = ll \begin{bmatrix} 0.5 & 0 \\ 0 & 0.1745(10^\circ) \end{bmatrix}
\end{aligned} \tag{4-12}
$$

Note that the magnitude order of position and angle is different. $Q_{t,j}$ is the covariance of a single measurement. The whole Q_t is a 6X6 diagonal matrix. We pick out one representative initial estimate $\mu_0 = \begin{pmatrix} -9.61, & 3.60, & -1.53 \end{pmatrix}^T$ to visualize. Both qualitative and quantitative results are shown below.



(a) Trajectory cost and path. In cost curve, RLC-EKF and EKF's cost are shown in green and red respectively. In path curve, RL estimation, EKF estimation, dead reconking path and ground truth are draw in green, red, black and blue solid lines respectively.



(b) Components of state vector x, y and θ tracking path

Figure 4-4: Qualitative result of localization task in moderate condition

The filled green and red area is the standard deviation since the model is executed 100 times, indicating the uncertainty caused by noise and dynamic landmark settings. The model is evaluated from random sampled initial estimates for quantitative results, indicating the RL outperforms EKF with confidence.

Table 4-2: RMSE of the localization task

RMSE	X[m]	Y[m]	[$\theta°$]	Total
RLC-EKF	0.689	0.193	0.086	0.779
EKF	0.91	0.343	0.115	1.087

The results show that RLC-EKF is better than EKF in terms of either total score or sub-items. Also, similar to what we concluded from the orientation estimation task, the main advantage occurs at the beginning stage. From the standard deviation curve, we notice that even the slowest RLC-EKF convergence is faster than EKF convergence.

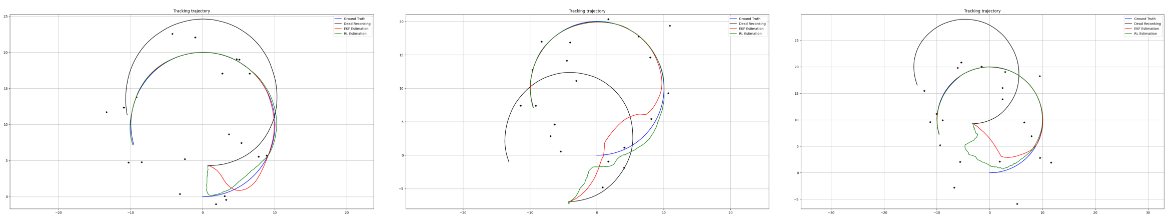
4-2-4 Initial Estimate

We continue to investigate the sweet point, initial estimate's effect on the performance of RLC-EKF. In this scenario, model covariance and measurement noise keep the same as the last subsection, while the policy is trained with the random initial estimates with the range of [-10m, 10m]. The trained model is then evaluated on four different initial ranges, and the results are shown below:

Table 4-3: RMSE with different initial deviation

Deviation range[m]	[-3, 3]	[-5, 5]	[-10, 10]	[-15, 15]
RLC-EKF	0.276	0.377	0.59	0.846
EKF	0.287	0.422	0.734	1.101
Advantage Ratio	1.039	1.119	1.244	1.301

The result table indicates the larger the initial deviation range, the more advantage our RLC-EKF agent can obtain. This proves the sweet point assumption. The model is trained with a deviation range of [-10m, 10m], but it still performs well in the range of [-15m, 15m]. Another notice is RLC-EKF that barely gains an advantage when the initial deviation is small ([-3m, 3m]). In this case, our algorithm has little difference from the pure EKF.

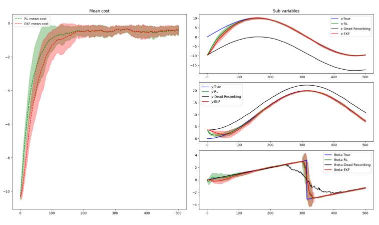


(a) $\mu_0 = 1.2, 4.7, 1.0$ (b) $\mu_0 = -4.8, -7.2, 0.5$ (c) $\mu_0 = -4.5, -8.1, 0.7$

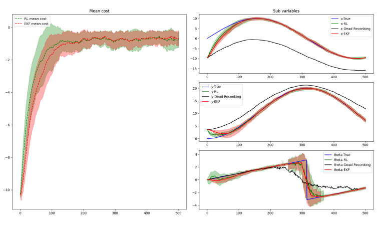
Figure 4-5: Representative trajectories with different initial estimates for visualization

4-2-5 Measurement Noise

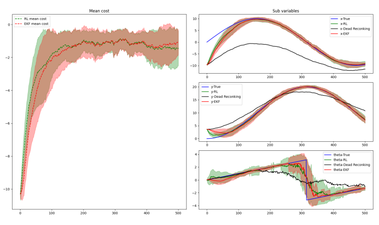
Measurement noise on both input data and sensor data affects the performance of the localization significantly. We boost the noise level used in (4-12) by **2**, **5**, and **10** times respectively. For benchmark purpose, we fixed the initial estimate the same as in section 4-2-3. The results are shown as follows:



(a) 2 times boost



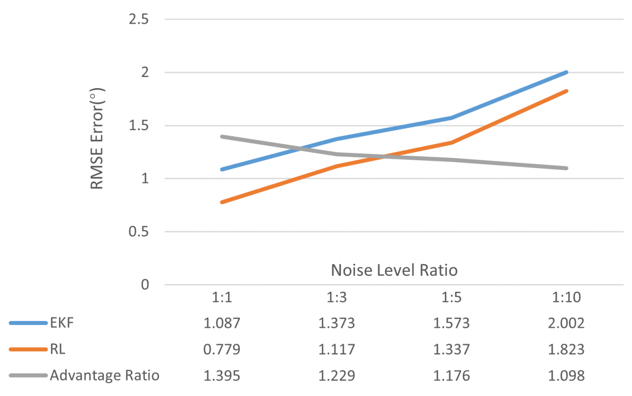
(b) 5 times boost



(c) 10 times boost

Figure 4-6: Qualitative results of localization with different measurement noise level. Both cost curve and variables in state vector are compared.

From the figures, we can observe, with the increase of the noise level, the uncertainty of both RLC-EKF and EKF's estimation expands during Monte Carlo Simulation. However, the uncertainty of RLC-EKF is always smaller than EKF, which means it's more robust within a high perturbation environment. The total cost and sub-items cost curves also show superior accuracy after RL compensation.



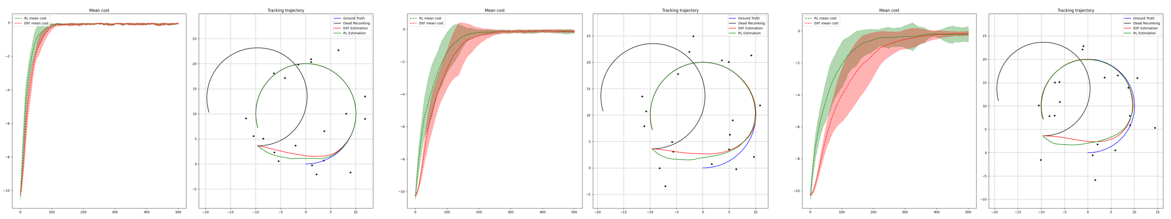
Noise Level Ratio	1:1	1:3	1:5	1:10
EKF	1.087	1.373	1.573	2.002
RL	0.779	1.117	1.337	1.823
Advantage Ratio	1.395	1.229	1.176	1.098

Figure 4-7: RMSE with different measurement noise level.

The result table indicates our advantage ratios go down when the noise level is higher. That conflicts with the results we obtained for orientation estimation in section 3-2-5. This may be because the original noise level is already high, and there is always a threshold for noise if we seek the most advantage.

4-2-6 Model Covariance

Model covariance actually has non-negligible relationship with the noise intensity. In practical applications, since both input and sensor measurement noises are unknown, we have to decide them by statistical computation. This noise level estimation is then reflected on EKF's model covariance on both process model and measurement model. We regard this as the calibration procedure of kalman filtering, for the specific experiment condition. The relative ratio between process model covariance R_t and measurement model Q_t, indicates which value do we trust more between internal odometry and external sensor data. The more the ratio, means the relative larger uncertainly on odometry data. So we trust on more measurement update and less on motion update. Since our RL agent only contributes to the measurement update, it holds the chance that we will enjoy more benefits when the ratio is larger. To validate this idea, we set the model covariance ratio in (4-12) as 1:1, and expand measurement model covariance by **0.5**, **2**, **4** times. The results are shown as follows:



(a) Ratio 1:0.5 (b) Ratio 1:2 (c) Ratio 1:4

Figure 4-8: Qualitative results of localization considering different model covariance ratio, the result of ratio 1:1 is shown in Figure 4-4.

With the increase of the covariance ratio, we start to trust the measurement update less. That's why the uncertainty increases, and the curve converges slower. Meanwhile, it leaves more space for the RL agent to compensate at the beginning stage. That's why the RL advantage expands.

Table 4-4: RMSE with different model covariance ratio

Covariance ratio	1: 0.5	1: 1	1: 2	1: 4
RLC-EKF	0.537	0.779	1.08	1.37
EKF	0.693	1.087	1.476	2.191
Advantage Ratio	1.29	1.395	1.366	1.599

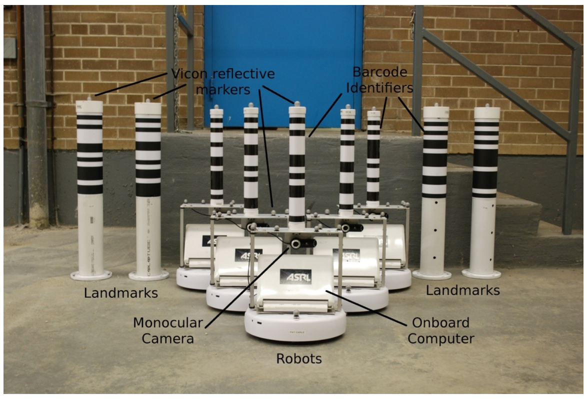
From the quantitative results, we say RLC-EKF always outperforms pure EKF for all model covariance ratios. This means if there is no estimator pre-calibration in advance, i.e., the ratio or quantity of the covariance is not known in advance, RLC-EKF will always do better, even with the same performance as the well-calibrated EKF. Plus, the more inappropriate the calibration is, the more potential advantages we can obtain from RL compensation.

4-3 Real Dataset

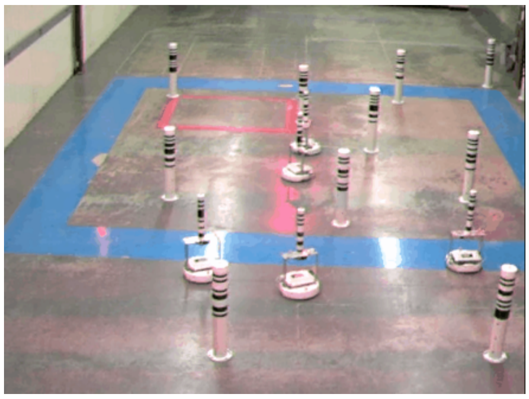
After the concise simulation experiments, we move forward to implement the proposed idea in a real application scenario. We didn't specify exact sensor types in the simulation stage as the proposed framework can actually be applied to different sensor combinations. For example, odometry data (velocity and angular velocity) can be provided by either IMU, chassis controller, or wheel encoder[27]. Meanwhile, range-bearing feature measurement can be provided by camera, laser range finder (LRF)[28], UWB or even radar[29]. This gives us more flexible options when searching for a real dataset. Eventually, since our main interest is on indoor localization scenarios, we choose the UTIAS[2] dataset published by the Autonomous Space Robotics Lab (ASRL) at the University of Toronto Institute for validating our algorithm.

4-3-1 UTIAS Dataset

UTIAS dataset is initially created for solving an indoor multi-robot cooperative localization and mapping problem. The dataset collection contains nine datasets in total. Each dataset has odometry and (range and bearing) measurement data from 5 robots, as well as accurate ground truth data for all robot poses and 15 landmark positions. Within a 15m X 8m laboratory field, five robots move with random waypoints while recording both ground truth and measurement data from both internal and external sensors. Each moving robot is based on the iRobot Create platform and equipped with an onboard computer and a monocular camera for sensing. There are 15 landmarks within the field, all cylindrical tubes with attached barcodes indicating their identity. Odometry data is directly logged from the robot's controller, while the robot's and landmark's groundtruth pose are obtained from a 10-camera Vicon motion capture system at 100Hz with accuracy on the order of $1X10^{-3}m$. For feature measurements, images captured by the onboard monocular camera with a resolution of 960×720, processed to detect the barcodes on landmarks and other robots. By solving the bounding boxed from camera frame to world coordinate and matching the barcode identities, range, bearing and correspondence can be obtained and formulated the measurement. Since our experiment is self-localization with a static map, we wipe the other four moving robots out. By only utilizing one robot's odometry data and its observations upon 15 other landmarks, the experiment is ready to go.



(a) Equipment included in experiment



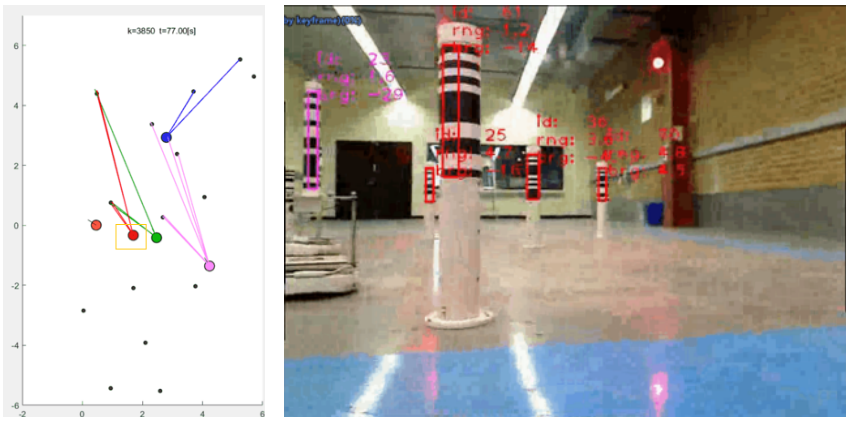
(b) Experiment field

Figure 4-9: Hardware essentials of UTIAS dataset[2]

4-3-2 Dataset processing

Dataset sampling

The timestamp in the original dataset is not aligned for different sources of data. There are official Matlab scripts for loading the dataset provided by the publisher. We utilize the toolkit to process the data based on our requirements, save it and load it in python to use. For time synchronization, we pick the sampling rate as 0.2s. The odometry data with the nearest timestamp becomes the value for that timestamp, while all available measurements within the sampling range are collected and stacked as the current measurements.



(a) Dataset visualization (b) Barcode detection process

Figure 4-10: Illustration of UTIAS dataset loading and camera sensing process. In figure 4-10(a), one dataset is loaded in Matlab, we only focus on one robot and filter out others

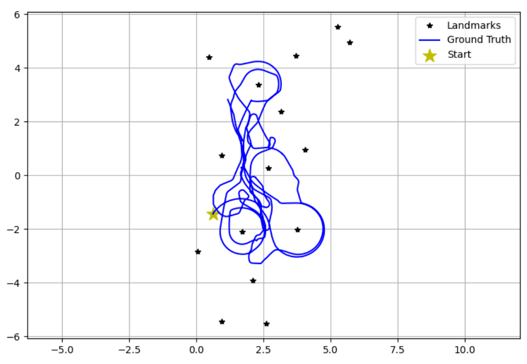
Outliner removal

To adapt our algorithm on the dataset, some outliners or unnecessary information are filtered out manually, they are:

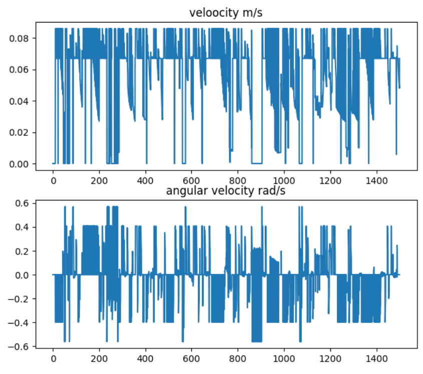
- Observation of other robots: These robots serve as dynamic landmarks, but it's not accepted in our algorithm since all landmarks are assumed static.
- Wrong barcode matching: When there is a problem with the barcode detection, the measured identity number cannot match the existing landmark, we choose to abandon these measurements.
- Absolute wrong measurements: We notice during the experiment that some measurements are out of the reasonable noise range and are obviously wrong. We filtered out these measurements by computing the cost between the measurements and its *should-have-observed* true value.

4-3-3 Train and Evaluation

The select dataset lasts for 1500s. During the operation time, the robot drives freely within the field, which is shown in figure 4-11.



(a) Ground truth trajectory



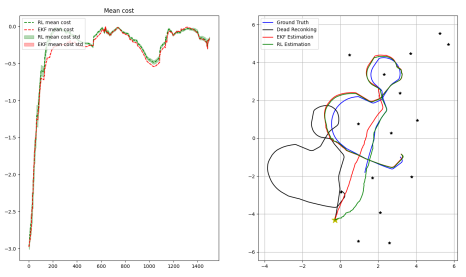
(b) Odometry input

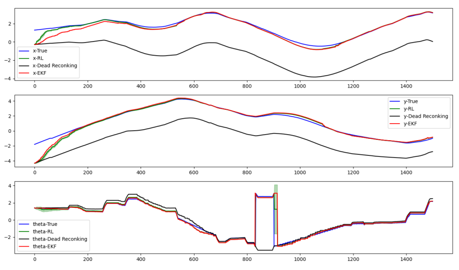
Figure 4-11: Data visualization of $robot5$ from the UTIAS $dataset - 1$

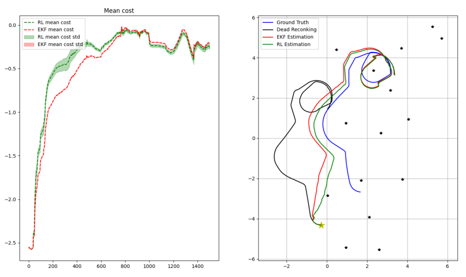
In practice, since not all landmarks are within the camera sensing view, the availability of the measurements is far more sparse than what we obtain in simulation experiments. In this case, we lower the update measurement number from **3** to **2**, which also adjusts our RL action space from 3X6 to 3X4. For the training process, we train the policy with the first 300s and evaluate on the rest of the trajectory. We evaluate the dataset on **Scenario 1: duration 400-700s**, **Scenario 2: duration 780-1080s** and **Scenario 3: whole rest trajectory**, three scenarios. Each dataset is trained with an individual policy respectively. The rest of the training settings are the same as table 4-1.

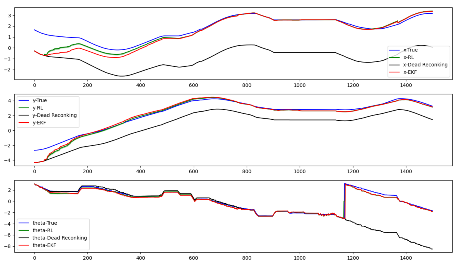
Table 4-5: RMSE with different model covariance ratio

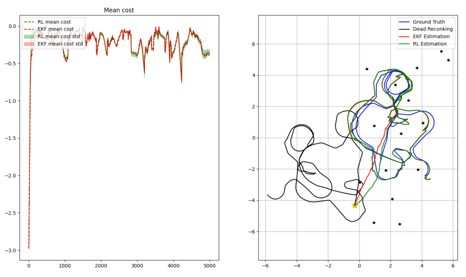
RMSE	X[m]	Y[m]	[θ°]	Total
		Scenario 1		
RLC-EKF	0.172	0.307	0.166	0.393
EKF	0.382	0.452	0.184	0.625
		Scenario 2		
RLC-EKF	0.278	0.224	0.163	0.377
EKF	0.369	0.242	0.177	0.472
		Scenario 3		
RLC-EKF	0.123	0.129	0.345	0.23
EKF	0.117	0.123	0.317	0.243


(a) Scenario 1


(b) Scenario 1: Sub items


(c) Scenario 2


(d) Scenario 2: Sub items


(e) Scenario 3

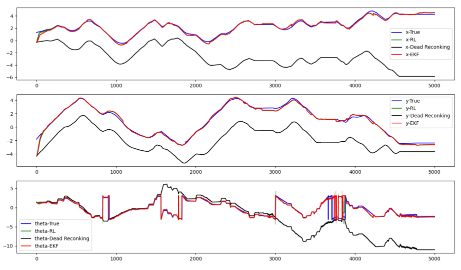

(f) Scenario 3: Sub items

Figure 4-12: Cost curve, trajectory tracking and variable tracking on three different evaluation scenarios.

From both qualitative and quantitative results, we can observe RLC-EKF has better performance than EKF on different test sets. Since the advantages in the early stage are relatively large, the performance on the whole trajectory does not outperform too much.

Chapter 5

Conclusion

5-1 Question Answer

After the theoretical derivation and experiments, we can finally answer the research questions asked at the beginning of this thesis:

What's the relationship between deep reinforcement learning and filtering-based state estimator such as EKF?

Both DRL and EKF hold the Markov assumption and propagate state recursively under the instruction of Bayes rule. With this premise, EKF can be modeled as a Markov Decision Process problem. We create an MLP formed policy network, which generates an end-to-end estimator gain matrix from the current error state input. In offline training stage, The error between estimation and nominal value serves as the environment reward. The policy is taught from such reward by continuously interacting with the environment. In online inference stage, with previously learned experience, the RL agent works in the same manner as EKF. The only difference is the estimator gain is generated by policy instead of formula computation.

How can reinforcement learning be combined with EKF for better pose estimation performance?

From the explanation in the last section, the RL agent can work solely as an online estimator, hopefully with the equivalent performance of EKF. Instead of utilizing the RL estimator standalone, we finally proposed a merged method that combines both EKF and RL's advantage. After a standard EKF, the RL agent conducts a second-time measurement update to compensate for the residual error between the EKF estimation and the ground truth value. As the policy is trained offline, RL knows how to give a proper estimator gain for mitigating the residual error based on the current error state.

How can the proposed reinforcement learning filter idea be applied to specific pose estimation tasks such as orientation estimation or localization?

In this thesis, we eventually implement the proposed RLC-EKF idea on two physical frameworks, solving two practical problems, i.e., orientation estimation and plane localization.

Both implementations are based on their existing standard EKF frameworks. Based on different task requirements, RL states are also set up accordingly (roll, yaw, pitch for orientation estimation, x, y, theta for localization). We also tried different RL training settings (hyperparameter tuning, episode number, reward function) specifically for each experiment to achieve the most plausible results. The idea applies to various practical applications, as long as there are already suitable EKF applications.

5-2 Discussion

In order to analyze the algorithm comprehensively, we conduct simulation experiments first. Then for both two tasks, we find suitable datasets that including actual sensors. From the results of both synthetic and real data, the pros and cons of the algorithm are finally summarized as follows:

Profits

- In orientation estimation experiments. RLC-EKF first outperforms EKF in normal estimation conditions. Then we validate the robustness by varying measurement noise, initial estimates, and angular velocity. The results lead to the conclusion that, the more noise disturbance and initial deviation, the more advantage we can obtain from RLC-EKF for orientation estimation compared to pure EKF. Also, we prove the success of using error dynamics as the RL state, as the model doesn't only recognize the angular velocity profile it trained from but all other profiles.
- In localization estimation experiments, same as orientation estimation, RLC-EKF is first proved to work in a moderate environment and then evaluate with variations. Unlike orientation estimation, we don't include a generality test for this scenario as the model is already trained and evaluated in a dynamic changing environment. To wrap up, the less measurement noise and larger initial deviation, the more advantage we can get from RLC-EKF for localization tasks. Another change we bring for the localization task is the variation of covariance, which represents well or bad sensor calibrations. The result implies no matter which calibration quality, RLC-EKF always has space to improve.
- In summary, since no matter which severe condition we choose, RLC-EKF always holds relatively large or small advantages than EKF, at least no worse than original EKF estimation, we can finally conclude that the proposed idea is indeed a useful assistant method that improves EKF on both accuracy and robustness aspects. We only testify the idea for two application fields, but the scope can be turned to other topics in future research.

Limitations

- We may notice in orientation estimation the advantage ratio increases with the measurement noise level, but in the localization task, the trend is opposite. The possible cause of this problem is the difference in noise magnitude order. For localization, maybe the noise level is relative too loud, so the RL agent is confused to relate the innovations with the appropriate estimator gain matrix, which degrades the performance. So for the best advantage, the measurement level should also be appropriately allocated.

- In this thesis, all RL states and actions are in continuous form. This actually brings more intensive numerical resolution than in discrete form and makes the training of the policy more difficult. Other problems occurred because of resolution lay on the initial deviation range and the selection of action space. During experiments, if we choose a too large initial range or inappropriate RL action space, the policy network will hardly converge. For successful training, such a variable determination process is always strenuous and not promising.
- Our RLC-EKF framework still lacks the ability to deal with the dynamic number of the measurements as the action space's shape is fixed in the RL method. In the localization task, we fixed to update two or three measures per step, with selection criteria. The waste of information actually affects the performance of both EKF and RLC-EKF.
- From cost curves, we also notice that RLC-EKF gains the most benefits at the beginning convergence stage. Such faster convergence makes us ignore the performance of RLC-EKF after stabilization. In case of small initial deviation or fast EKF convergence, the existence of RL seems dispensable. That's reasonable since the improvement space is already tiny enough after stabilization. Still, this point is worth mentioning so we can decide the suitable application condition of our algorithm rationally.
- In practical applications, there are already different tricks narrowing down the initial estimate and calibrating the measurement noise. So the algorithm's objective use-value in industries is yet to be considered.

5-3 Future works

In this section, we suggest several next step topics to explore. We come up with these ideas during conducting the thesis.

State expansion

In this thesis, the state vector components for both experiments are selected as the most basic ones, as the idea is still immature. In the future, more valuable information can be added and estimated together for better performance, e.g., velocity, acceleration for localization task, and gyroscope bias for orientation estimation.

6 DOF pose estimation

Initially, we intend to implement the idea directly on a 6 DOF pose estimation task after finishing the orientation estimation experiment. The task is very common in the drone research area. There are lots of mature visual-inertial odometry frameworks such as MSCKF[30] and ROVIO[31]. We also tried to implement the idea on MSCKF framework, which turns to be totally infeasible. Such algorithms usually include environment landmarks into the state vector, either for localization or the entire SLAM. For example, MSCKF needs a state augmentation step for tracking observations. Such setting forces our RL agent to neglect other info in state vector, but update poses only since the state vector size of the RL agent is fixed.

Another problem is the map in 6DOF space is a far more complex structure than 3DOF. It even takes massive learning efforts to save the map, which is unsuitable for a one-year thesis project. In the future, with no time restrictions, such an idea of 6DOF pose estimation, as well as the SLAM topics, can be further explored.

Multi-robot cooperative localization

In the UTIAS dataset, we reject the other four robots and only use one robot for self-localization. Don't neglect this dataset is initially created for robot cooperative localization. In this case, if each robot is treated as an RL agent, a multi-agent reinforcement learning problem can be formulated. Based on the game theory, five robots can work together to maximize the overall localization accuracy. Such an exciting idea is a worthy exploration in the future.

Appendix A

Supplementary Experiments

In this appendix some supplementary experiments we done besides the main topic, as well as some RL model training details are given.

A-1 Direct RL estimator

The main topic of this thesis is the RL compensated EKF non-linear estimator, which is derived from pure RL estimator idea which is already introduced in 2-3. Pure RL estimator replaces EKF's measurement update at all, meaning there is only one measurement update per step and the estimator gain is computed by RL policy. Before implementing the new compensated idea, we first replicate a pure RL estimator for orientation estimation task which is invented in [32].

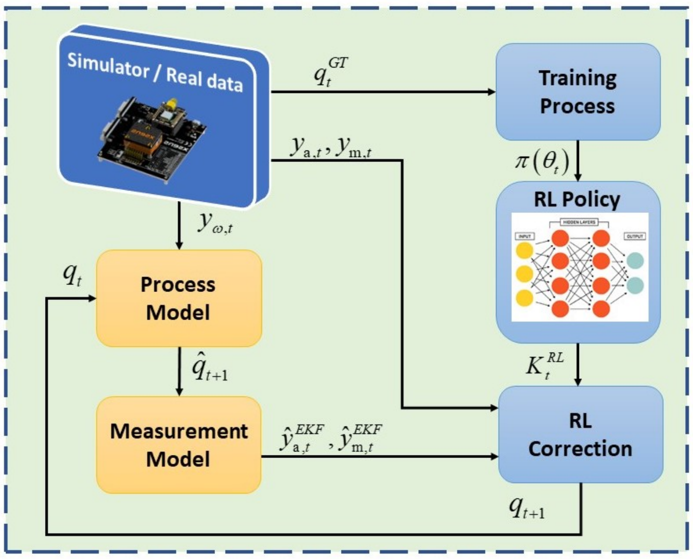
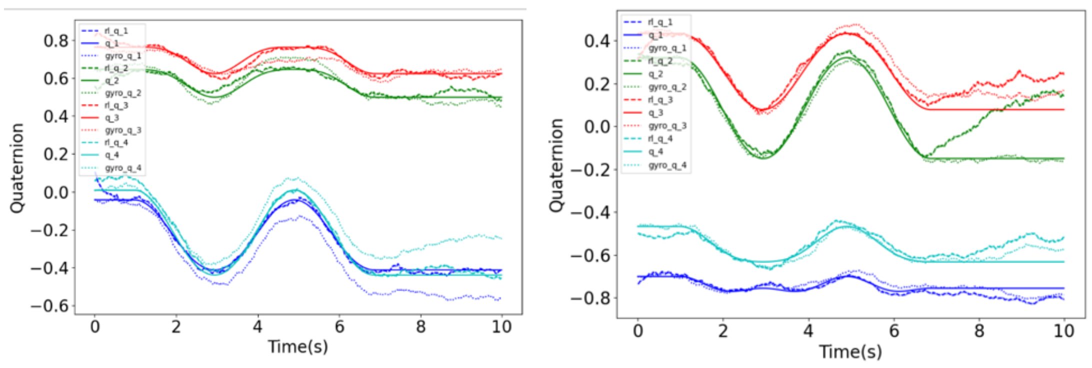


Figure A-1: Direct RL estimator for orientation estimation

We keep the same training settings as in table 3-1. The results by the best model training from PPO2 is shown in figure A-3. However, the RL estimator doesn't provide stable tracking for different noise and initial deviation settings. For example in figure A-2(b), RL estimator loses the quaternion tracking at the turning point.



(a) Success estimation (b) Failed estimation

Figure A-2: Tracking performance of direct RL estimator. We compare the result between gyroscope update(dotted line) and RL update(dashed line).

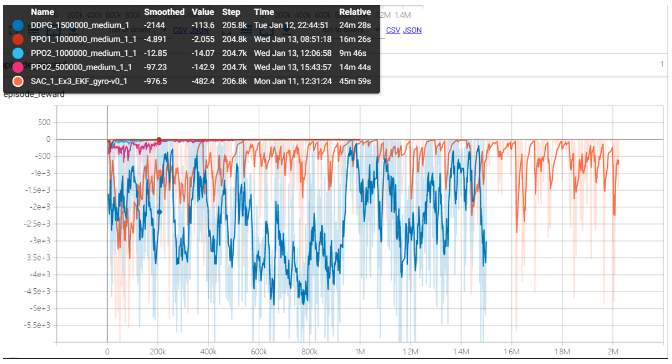
From the quantitative results we notice the direct RL estimator gives comparable performance than EKF when in case of not diverging. However, because of occasional divergence, the overall performance is with great randomness, which makes the algorithm less robust in changing environments. That's why we propose RLC-EKF for more guaranteed tracking. As the RL state is set the same as current quaternion, instead of error state, the potential cause of this problem is maybe the sudden change of the state.

A-2 Training details

In this section we roughly explain how we decide the best RL algorithm and correspondent hyperparameters setting that are used for our RL model.

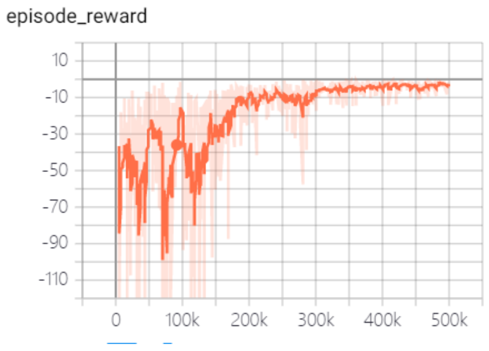
RL algorithm selection

In stable-baseline, there are several mainstream RL algorithms with standard implementation and integration. They are either policy based, value based, or merged DRL algorithms. At the beginning of the training stage, we tried SAC (soft Actor-Critic)[33], DDPG[34], PPO1 and PPO2 algorithms from the stable-baseline library, with default parameter settings. By evaluating the training loss curve in tensorboard panel as shown in figure A-3(a), we rejected SAC and DDPG since they don't converge during the most of time. Since ppo2 is an upgraded version of ppo1 and supports parallel environment training, which is faster, we finally choose PPO2 to train our agent.



	Name	Smoothed	Value	Step	Time	Relative
●	DDPG_1500000_medium_1	-2144	-113.6	205.8k	Tue Jan 12, 22:44:51	24m 28s
●	PPO1_1000000_medium_1_1	-4.891	-2.055	204.8k	Wed Jan 13, 08:51:18	16m 26s
●	PPO2_1000000_medium_1_1	-12.85	-14.07	204.7k	Wed Jan 13, 12:06:58	9m 46s
●	PPO2_500000_medium_1_1	-97.23	-142.9	204.7k	Wed Jan 13, 15:43:57	14m 44s
●	SAC_1_Ex3_EKF_gyro-v0_1	-976.5	-482.4	206.8k	Mon Jan 11, 12:31:24	45m 59s

(a) Different RL algorithms comparison



(b) PPO2 solely

Figure A-3: Training loss of RL algorithms reading from tensorboard log.

After fixing the PPO2, we also varying other hyperparameters to obtain the best trained RL policy model. The variations are:

- Different learning rates: $[0.00025, 0.0025, 0.025]$
- Gammaγ of PPO: $[0.99, 0.3, 0.25]$
- Number of episodes: $[100, 300, 500, 1000]$
- Hidden dimension of MLP: $[64, 64], [64, 128], [128, 128], [128, 64, 32]$
- Action space range: $[-0.01, 0.01], [-0.02, 0.02], [-0.03, 0.03], [-0.05, 0.05], [-0.1, 0.1]$

Take the item *Number of episodes* for example. By varying this variable we intend to select the appropriate total training time. If the value is too small, the loss of RL has not converged to the minimum, and the RL model is too large, there is a risk of overfitting. By observing figure A-3(b) we find out the PPO2 model usually converges after 300 episodes. We train the model slightly longer so finally the number 500 is used for training.

In stale-baseline, they also offer a hyperparameter optimization tool based on **Optuna**[35], which can automatically decides a proper hyperparameter set for the training task. We haven't explored this toolbox but it's always nice to give a try in the future.

Appendix B

Code Description

In the spirit of sharing and hopefully making my little contribution for the future research, the source code of both orientation estimation and localization experiments are shared on Github. The code are available at:

https://github.com/Mrhamsterleo/RLC-EKF-Orientation-Estimation
https://github.com/Mrhamsterleo/RLC-EKF-Localization

Here we list the instruction of using RLC-EKF for localization experiment. The tutorial for orientation estimation just has slight difference.

B-1 Environment Setup

Clone the project

```
git clone git@github.com:Mrhamsterleo/RLC-EKF-Localization.git
cd RLC-EKF-Localization
```

It's always wise to create a virtual conda environment without affacting other projects. To create a new environemnt:

```
conda create -n RLC-EKF python=python=3.6.12
conda activate RLC-EKF
```

Install dependencies

After you created and activated the Conda environment, you have to install the python dependencies. This can be done using the following command:

```
pip install -r requirements.txt
```

B-2 Folder Structure

- *RLestimator_ekf.py*: Gym environment for RLC-EKF orintation estimator to conduct simulation experiments.
- *RLestimator_real_data.py*: Gym environment for RLC-EKF orintation estimator to conduct real dataset experiments.
- *train_evaluate.py*: Train and evaluate RL policy using stable-baseline platform. For evaluation, both qualitive and quantitive results compared to pure EKF performance can be provided.policy using stable-baseline platform.
- *train_evaluate_eva_list.py*: Evaluate a list of different policies (usually 10 models for a single training). The best model can be selected out from results.
- *data_simulator.py*: Generate range-bearing measurements, odometry input, ground truth of robot and landmarks, for simulation experiments.
- *load_dataset.py*: Data loader especially made for UTIAS dataset.

B-3 Training

For simulation related settings, in *RLestimator_ekf.py* change the following parameters.

Measurement noise

```
#line 104 (for odometry input noise)
self.Q_sim = np.diag([0.2, np.deg2rad(1.0)]) ** 2 * 0.1
#line 106 (for feature measurement noise)
self.R_sim = np.diag([1.0, np.deg2rad(10.0)]) ** 2 * 0.1
```

Initial deviation

```
#line 90 (range of uniform distribution)
self.initial_bias = np.random.uniform(-1, 1, size=(self.STATE_SIZE, 1)) * 3
```

Model covariance

```
#line 93 (for EKF process model covariance)
self.Cx = np.diag([1, 1, np.deg2rad(30.0)]) ** 2
#line 97 (for EKF measurement model covariance)
self.Cx_obs = np.diag([0.5, 0.5, 0.5, 0.5, 0.5, 0.5]) ** 2
```

Tracking trajectory

```
#line 110-111 (defined by velocities)
v = 1.0  # [m/s]
yaw_rate = 0.1  # [rad/s]
```

Landmark

Position devation for landmark placed in circular only, the landmark distribution can also be changed to other shape.

```
#line 136
m_sim = np.diag([2.5, 2.5]) ** 2
```

Other settings adjustment can be done in *train_evaluate.py*.

- *model_num*: Policy model you want to train for each script execution.
- *episodes*: The number of episodes for agent to explore.
- T: Total length of each eposide.
- *env_name*: The name of the gym formed enviroment.

To change the structure and activation of the MLP network, you can do:

```
# Custom MLP policy of two layers of size 32 each with tanh activation function
policy_kwargs = dict(act_fun=tf.nn.tanh, net_arch=[32, 32])
# Create the agent
model = PPO2("MlpPolicy", "Env_name", policy_kwargs=policy_kwargs, verbose=1)
```

When all settings are done, directly execute the script and wait for finishing. During the training, a tensorboard log file is generated in the folder ./logs. You can want the simutanous training loss through:

```
TensorBoard --logdir=logs/your_saved_model_name/PPO2_0_1 --host=localhost
```

B-4 Evaluation

Besides the experiment condition setting which is introduced in last section, there are few adjustments we can made for evaluation:

- *model = PPO2.load(”model_path”)*: To load the best trained model for agent to explore.
- *num_of_paths*: The number of Monte Carlo Simulation for stastical results neglecting odd disturbance.
- *MODEL_PATH*: Folder path of multiple models we want to select the best model from. After setting is done, we can directly run the evaluation script, and obtain cost, trajectory and sub-items graphs. The numerical results of EKF and RLC-EKF are also given in the terminal.
- *show_animation*: Whether display the result statically or simutanously.

Evaluation with real data

There are 4 types of UTIAS dataset with different sampling rate, presented in .mat format under the folder /datasets. For switching differnt dataset collection, change the line in "*RLestimator_ekf_realdata.py*":

```
#line 87-88 (robot label indicates individual robot you want to use)
dataset_path = "dataset/" + "MRCLAM0.2.mat"
robot_label = 5
```

Appendix C

Quaternion Basis

In this appendix we introduced quaternion's definition and mathematical operation involved in this thesis. They are summarized from book "Quaternion kinematics for the error-state Kalman filter[36]."

C-1 Definition

A quaternion can be represented as the combination of scalar and imaginary parts.

$$\begin{gathered} Q = q_w + q_x i + q_y j + q_z k \quad \Leftrightarrow \quad Q = q_w + \mathbf{q}_v \\ Q = \langle q_w, \mathbf{q}_v \rangle \\ \mathbf{q} \triangleq \begin{bmatrix} q_w \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \end{gathered} \tag{C-1}$$

By taking the notation $\{q_w, q_x, q_y, q_z\}$, the quaternion is presented in a 4X1 vector.

C-2 Quaternion operations

Sum

Quaternion sums by simply adds the scalar part and imaginary part respectively.

$$\mathbf{p} \pm \mathbf{q} = \begin{bmatrix} p_w \\ \mathbf{p}_v \end{bmatrix} \pm \begin{bmatrix} q_w \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} p_w \pm q_w \\ \mathbf{p}_v \pm \mathbf{q}_v \end{bmatrix} \tag{C-2}$$

Product Product is calculated by:

$$\mathbf{p} \odot \mathbf{q} = \begin{bmatrix} p_w q_w - \mathbf{p}_v^\top \mathbf{q}_v \\ p_w \mathbf{q}_v + q_w \mathbf{p}_v + \mathbf{p}_v \times \mathbf{q}_v \end{bmatrix} \tag{C-3}$$

In this thesis, instead of using the direct quaternion product formula, we utilize its two equivalent matrix products, namely left- multiply and right- multiply, given as:

$$\mathbf{q}_1 \odot \mathbf{q}_2 = [\mathbf{q}_1]_L \, \mathbf{q}_2 \quad \text{and} \quad \mathbf{q}_1 \odot \mathbf{q}_2 = [\mathbf{q}_2]_R \, \mathbf{q}_1$$
$$[\mathbf{q}]_L = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix}, \quad [\mathbf{q}]_R = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & q_z & -q_y \\ q_y & -q_z & q_w & q_x \\ q_z & q_y & -q_x & q_w \end{bmatrix} \tag{C-4}$$

Sometime we also use the cross-product inspection which involves *skew operator* $[\cdot]_x$:

$$\mathbf{q}t_L = q_w \mathbf{I} + \begin{bmatrix} 0 & -\mathbf{q}_v^\top \\ \mathbf{q}_v & [\mathbf{q}_v]_x \end{bmatrix}, \quad [\mathbf{q}]_R = q_w \mathbf{I} + \begin{bmatrix} 0 & -\mathbf{q}_v^\top \\ \mathbf{q}_v & -[\mathbf{q}_v]_\times \end{bmatrix} \tag{C-5}$$
$$[\mathrm{a}]_\times \triangleq \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

Conjugation

The conjugate of a quaternion is defined by:

$$\mathbf{q}^* \triangleq q_w - \mathbf{q}_v = \begin{bmatrix} q_w \\ -\mathbf{q}_v \end{bmatrix} \tag{C-6}$$

With properties,

$$\mathbf{q} \odot \mathbf{q}^* = \mathbf{q}^* \odot \mathbf{q} = q_w^2 + q_x^2 + q_y^2 + q_z^2 = \begin{bmatrix} q_w^2 + q_x^2 + q_y^2 + q_z^2 \\ 0_v \end{bmatrix}$$
$$(\mathbf{p} \odot \mathbf{q})^* = \mathbf{q}^* \odot \mathbf{p}^* \tag{C-7}$$

Norm Norm of a quaternion is used in this thesis for unit re-normalization after each quaternion update, it's simply the mean square root of 4 components in the quaternion vector.

$$\|\mathbf{q}\| \triangleq \sqrt{\mathbf{q} \odot \mathbf{q}^*} = \sqrt{\mathbf{q}^* \odot \mathbf{q}} = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} \in \mathbb{R} \tag{C-8}$$

For unit quaternions, $\|\mathrm{q}\| = 1$, and therefore,

$$q^{-1} = q^* \tag{C-9}$$

Exponential

We have, for $\mathbf{q} \in \mathbb{H}$ and $t \in \mathbb{R}$

$$\mathbf{q}^t = \exp\left(\log\left(\mathbf{q}^t\right)\right) = \exp(t \log(\mathbf{q})) \tag{C-10}$$

If $\|q\| = 1$, we can write $\mathbf{q} = [\cos\theta, \mathbf{u}\sin\theta]$, thus $\log(\mathbf{q}) = \mathbf{u}\theta$, which gives

$$\mathbf{q}^t = \exp(t\mathbf{u}\theta) = \begin{bmatrix} \cos t\theta \\ \mathbf{u}\sin t\theta \end{bmatrix} \tag{C-11}$$

Logarithm of unit quaternions

For unit quaternion, we have

$$\log\mathbf{q} = \log(\cos\theta + \mathbf{u}\sin\theta) = \log\left(e^{\mathbf{u}\theta}\right) = \mathbf{u}\theta = \begin{bmatrix} 0 \\ \mathbf{u}\theta \end{bmatrix} \tag{C-12}$$

C-3 Relation Between Euler Angle and Quaternion

We directly cite the very nice and clear table in [36] for the conversion between quaternion and Euler angle.

	Rotation matrix, $\mathbf{R}$	Quaternion, q
Parameters	$3 \times 3 = 9$	$1 + 3 = 4$
Degrees of freedom	3	3
Constraints	$9 - 3 = 6$	$4 - 3 = 1$
Constraints	$\mathbf{R}\mathbf{R}^\top = \mathbf{I}; \det(\mathbf{R}) = +1$	$\mathbf{q} \otimes \mathbf{q}^* = 1$
ODE	$\dot{\mathbf{R}} = \mathbf{R}[\boldsymbol{\omega}]_\times$	$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \boldsymbol{\omega}$
Exponential map	$\mathbf{R} = \exp\left([\mathbf{u}\phi]_\times\right)$	$\mathbf{q} = \exp(\mathbf{u}\phi/2)$
Logarithmic map	$\log(\mathbf{R}) = [\mathbf{u}\phi]_\times$	$\log(\mathbf{q}) = \mathbf{u}\phi/2$
Relation to $SO(3)$	Single cover	Double cover
Identity	$\mathbf{I}$	1
Inverse	$\mathbf{R}^\top$	$\mathbf{q}^*$
Composition	$\mathbf{R}_1\mathbf{R}_2$	$\mathbf{q}_1 \otimes \mathbf{q}_2$
Rotation operator	$\mathbf{R} = \mathbf{I} + \sin\phi[\mathbf{u}]_\times + (1 - \cos\phi)[\mathbf{u}]_\times^2$	$\mathbf{q} = \cos\phi/2 + \mathbf{u}\sin\phi/2$
Rotation action	$\mathbf{R}\mathbf{x}$	$\mathbf{q} \otimes \mathbf{x} \otimes \mathbf{q}^*$
Interpolation	$\mathbf{R}^t = \mathbf{I} + \sin t\phi[\mathbf{u}]_\times + (1 - \cos t\phi)[\mathbf{u}]_\times^2$	$\mathbf{q}^t = \cos t\phi/2 + \mathbf{u}\sin t\phi/2$
	$\mathbf{R}_1\left(\mathbf{R}_1^\top\mathbf{R}_2\right)^t$	$\mathbf{q}_1 \otimes (\mathbf{q}_1^* \otimes \mathbf{q}_2)^t$
		$\mathrm{q}_1\frac{\sin((1-t)\Delta\theta)}{\sin(\Delta\theta)} + \mathbf{q}_2\frac{\sin(t\Delta\theta)}{\sin(\Delta\theta)}$

(C-13)

Cross relations

$$\begin{aligned}
\mathbf{R}\{\mathbf{q}\} &= \left(q_w^2 - \mathbf{q}_v^\top\mathbf{q}_v\right)\mathbf{I} + 2\mathbf{q}_v\mathbf{q}_v^\top + 2q_w\left[\mathbf{q}_v\right]_\times && \\
\mathbf{R}\{-\mathbf{q}\} &= \mathbf{R}\{\mathbf{q}\} && \text{double cover} \\
\mathbf{R}\{1\} &= \mathbf{I} && \text{identity} \\
\mathbf{R}\left\{\mathbf{q}^*\right\} &= \mathbf{R}\{\mathbf{q}\}^\top && \text{inverse} \\
\mathrm{R}\left\{\mathbf{q}_1 \otimes \mathbf{q}_2\right\} &= \mathbf{R}\left\{\mathbf{q}_1\right\}\mathbf{R}\left\{\mathbf{q}_2\right\} && \text{composition} \\
\mathbf{R}\left\{\mathbf{q}^t\right\} &= \mathbf{R}\{\mathbf{q}\}^t && \text{interpolation}
\end{aligned} \tag{C-14}$$

Bibliography

[1] D. Roetenberg, H. J. Luinge, C. T. Baten, and P. H. Veltink, "Compensation of magnetic disturbances improves inertial and magnetic sensing of human body segment orientation," *IEEE Transactions on neural systems and rehabilitation engineering*, vol. 13, no. 3, pp. 395–405, 2005.

[2] K. Y. Leung, Y. Halpern, T. D. Barfoot, and H. H. Liu, "The utias multi-robot cooperative localization and mapping dataset," *The International Journal of Robotics Research*, vol. 30, no. 8, pp. 969–974, 2011.

[3] P. Srinivas and A. Kumar, "Overview of architecture for gps-ins integration," in *2017 Recent Developments in Control, Automation & Power Engineering (RDCAPE)*, pp. 433–438, IEEE, 2017.

[4] S. A. Mohamed, M.-H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, "A survey on odometry for autonomous navigation systems," *IEEE Access*, vol. 7, pp. 97466–97486, 2019.

[5] D. Titterton, J. L. Weston, and J. Weston, *Strapdown inertial navigation technology*, vol. 17. IET, 2004.

[6] Y. S. Suh, "Orientation estimation using a quaternion-based indirect kalman filter with adaptive estimation of external acceleration," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 12, pp. 3296–3305, 2010.

[7] A. Gelb, *Applied optimal estimation.* MIT press, 1974.

[8] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[9] C. Luo, X. Chu, and A. Yuille, "Orinet: A fully convolutional network for 3d human pose estimation," *arXiv preprint arXiv:1811.04989*, 2018.

[10] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni, "Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[11] C. Chen, B. Wang, C. X. Lu, N. Trigoni, and A. Markham, "A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence," *arXiv preprint arXiv:2006.12567*, 2020.

[12] J. Morimoto and K. Doya, "Reinforcement learning state estimator," *Neural computation*, vol. 19, no. 3, pp. 730–756, 2007.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[14] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[17] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[19] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[20] L. Hu, C. Wu, and W. Pan, "Lyapunov-based reinforcement learning state estimator," *arXiv preprint arXiv:2010.13529*, 2020.

[21] M. Kok, J. D. Hol, and T. B. Schön, "Using inertial sensors for position and orientation estimation," *arXiv preprint arXiv:1704.06053*, 2017.

[22] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3." `https://github.com/DLR-RM/stable-baselines3`, 2019.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[24] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[25] S. B. Samsuri, H. Zamzuri, M. A. A. Rahman, S. A. Mazlan, and A. Rahman, "Computational cost analysis of extended kalman filter in simultaneous localization and mapping (ekf-slam) problem for autonomous vehicle," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 17, pp. 153–158, 2015.

[26] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009. revision #137311.

[27] Y. Cheng, M. Maimone, and L. Matthies, "Visual odometry on the mars exploration rovers," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 903–910, IEEE, 2005.

[28] Y. Zhuang, S. Yang, X. Li, and W. Wang, "3d-laser-based visual odometry for autonomous mobile robot in outdoor environments," in *2011 3rd International Conference on Awareness Science and Technology (iCAST)*, pp. 133–138, IEEE, 2011.

[29] R. Ghabcheloo and S. Siddiqui, "Complete odometry estimation of a vehicle using single automotive radar and a gyroscope," in *2018 26th Mediterranean Conference on Control and Automation (MED)*, pp. 855–860, IEEE, 2018.

[30] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3565–3572, IEEE, 2007.

[31] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 298–304, IEEE, 2015.

[32] L. Hu, Y. Tang, Z. Zhou, and W. Pan, "Reinforcement learning for orientation estimation using inertial sensors with performance guarantee," *arXiv preprint arXiv:2103.02357*, 2021.

[33] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[35] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

[36] J. Sola, "Quaternion kinematics for the error-state kalman filter," *arXiv preprint arXiv:1711.02508*, 2017.