Bayesian inference of travel paths from sporadic low-power radio beacon observations

Mantas Pajarskas

University of Oxford

Trinity 2017

Project supervisor:
Prof. Alex Rogers

Abstract

Low-power, small-scale Bluetooth radio beacons provide a novel approach for tracking animals in their natural habitat. As a specific case, flight path prediction of honey bees is investigated, using a bee-mounted radio beacon and an array of ground based sensors. We explore different machine learning approaches for the travel path inference, specifically comparing discrete hidden Markov models with continuous Gaussian processes. Both models are empirically compared in simulation, and then a prototype bee tracking system is implemented using hidden Markov models.

Employing domain-specific optimisations and low-level concurrency controls, we manage to create a parallelised, extensible and efficient bee tracking software which can simultaneously track hundreds of bees even on the low-resource machine. We extend the system provided by the project supervisor to incorporate the prediction engine and enhance the visualisation of the flight path, resulting in a prototype system ready to be deployed in the field.

Finally, we provide theoretical lower bounds on the prediction accuracy using Bayesian Cramér-Rao bounds to evaluate the performance of the model compared to the information-theoretic limit.

Contents

1 Introduction

1.1 Motivation

The problem of tracking the animal behaviour and movement patterns is of particular importance in biology and ecology (Levin 1992). Advances in tracking technology allows novel insights on animals navigation, as noted by Reynolds et al. (2007) for foraging bees and by Mann et al. (2011) for homing pigeons. Tracking insects and other small animals presents unique challenges due to the limited hardware size. Common approaches include miniaturised radio transmitters and harmonic radars, used by Wikelski et al. (2010) and Wolf et al. (2014).

1.2 Challenges

We investigate the usage of compact and low-power Bluetooth radio beacons for tracking the behaviour of animals in their natural environment. Such a system would use ground based sensors to receive the beacon ID and record the signal strength. As a concrete use case, we turn our attention to the problem of tracking the flight paths of honey bees (*Apis mellifera*).

This presents several challenges: firstly, the size of the bee limits the size of the Bluetooth transmitter and a battery, facilitating the need to conserve power as much as possible, hence, we can only send a signal occasionally. Secondly, in order for the cost of this approach to be feasible, we cannot afford to have too many ground sensors, hence, they will have to be placed on the ground sparsely. Furthermore, we aim to track a large number of bees simultaneously, thus the approach must be efficient.

1.3 Requirements

The goal of this project is to empirically select the best machine learning techniques and build a prototype bee tracking software.

We identify several crucial requirements for the system:

(R1) To conserve power, we must only occasionally send a radio signal, hence, we will aim to maximise the delay between observations without sacrificing accuracy.

(R2) To keep the cost reasonable, the number of ground sensors has to be limited and thus they can only be placed on the ground sparsely.

(R3) The bee tracking software should provide real-time predictions of the bee flight paths. Furthermore, it should be possible to track multiple

bees simultaneously and the information should be displayed on a map in an intuitive format.

(R4) The bee tracking software should work without access to any server or cloud service. The software is intended to be run on a laptop in the actual field near the beehive, where we expect the internet connection to be slow or non-existent, so ideally internet should only be required to load the map. Also, it must be efficient: it must run smoothly even on low-resource machine and use a limited amount of memory.

(R5) Finally, tracking should be accurate. The main measure of accuracy throughout this report will be *root mean square error*. More precisely, if our predictions are $(\hat{x}_1, \hat{y}_1), \ldots, (\hat{x}_n, \hat{y}_n)$ and the bee is actually in coordinates $(x_1, y_1), \ldots, (x_n, y_n)$, the root mean square error is

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}-x)^2 + (\hat{y}-y)^2} \tag{1}$$

Also, we want the accuracy to be not far from the theoretical limit to make sure the accuracy is not far from optimal.

1.4 Our contributions

We develop a bee flight path simulator, including the simulation of the radars, based on the biologically sound model of the beehive. Using this simulator as a generator for both training and testing data, we apply two machine learning approaches to the problem: Hidden Markov models and Gaussian Processes, empirically selecting the best hyperparameters. We research and implement optimisations for these approaches to be feasible, including novel ideas for Hidden Markov models enabled by domain knowledge. Further, to compute information-theoretic bounds on prediction accuracy, we use an approximation technique called kernel density estimation, enabling us to apply the Bayesian Cramér-Rao bounds to our problem.

Finally, we select Hidden Markov models as the best approach and develop bee tracking software, building on the skeleton implementation provided by project supervisor. Our contributions include lightweight, highly optimised prediction module in C++ employing multithreading and intuitive GUI for predicted path visualisation. The system is efficient enough to track hundreds of bees in real time. This project is sponsored by Syngenta and the tracking software will be demonstrated in summer 2017 at Jealott's Hill International Research Centre.

1.5 Methods

We will be comparing two approaches commonly used in the machine learning field for tracking problems: *Hidden Markov models* and *Gaussian processes*. We will be interested of two tracking modes: filtering and smoothing. Filtering refers to the situation where data is coming datapoint-by-datapoint and predictions are made in real-time. Smoothing is the situation where all the observation data is known in advance, and location is predicted with knowledge of past and future observations.

Hidden Markov model (HMM) presupposes that we have a system with changing state (in this case, the location of the bee), and information about the state is available only through some observations. Crucially, this approach relies on the assumption that the probability of the next state depends only on the previous state. One caveat of using hidden Markov models is the need for states to be discrete – we will divide the map into square cells, each of which constitutes a state. We will refer to this process as *discretisation*. Furthermore, we note that hidden Markov models can benefit from the domain-specific optimisations, as noted in Murphy (2012, p. 777).

Gaussian processes (GPs) approach is conceptually different, since it models a distribution over *functions*. For a finite dataset it is enough to model a distribution over the values of a function at the finite set of points. Gaussian processes are inherently continuous and therefore suitable for representation of bees' movements through continuous space. Continuity also means that it is possible to get the estimate of the bee position at any time (not only once an observation has occurred) and naturally deal with any missing data. Moreover, each prediction is accompanied by variance at that point, thus quantifying the *uncertainty* about the predictions, which is essential for monitoring the performance of the model.

In order to understand how close the accuracy is to the theoretical limit, we will employ *Bayesian Cramér-Rao* bounds.

Cramér-Rao bounds is a tool commonly used in statistics to give a lower bound for the mean square error for *deterministic* parameters (Schervish 1997), subject to various regularity conditions. Van Trees (1968) derived the analogous bound for *random variables*, called Bayesian Cramér-Rao bound. However, this approach requires inverting a huge matrix (scaling quadratically in number of observations), making it infeasible for our purposes. We will use a refinement of this technique (Dauwels 2005), making the computation tractable.

One of the key parts of the Bayesian Cramér-Rao bound computation requires knowledge of the probability density function (pdf) of bee location and observations. Given the complexity of the bee simulation algorithm, no

analytic form can be given. Thus, we will resort to estimating the pdf by the *kernel density estimation* method, effectively representing the density by mixture of Gaussian densities.

1.6 Structure of the report

The rest of the report is structured as follows: the background for the techniques used is presented in Section 2. The modelling process is described in Section 3. The in-depth accuracy comparison of the models is presented in Section 4, and in Section 5 we discuss the implementation of bee tracking software. In Section 6 we calculate the approximate lower bounds and compare them to our practical results. Finally, in Section 7, we present the conclusions, limitations and possible extensions of the project.

2 Background

We will briefly review the background knowledge on Hidden Markov models, Gaussian Processes, Bayesian Cramér-Rao bounds and kernel density estimation necessary to understand the project.

2.1 Hidden Markov models

In the Hidden Markov model framework, we identify *hidden states:* latent random variables which are not observable (denoted $\boldsymbol{z}_1, \boldsymbol{z}_2, \dots$), each associated with an corresponding random variable $\boldsymbol{x}_i$, called an *observation.*

We will model the actual flight path of a honey bee as a sequence of latent variables $\{\boldsymbol{z}_i\}_{i\geq 1}$ and we will denote the radio beacon observations (the location of the ground station and the signal strength) as corresponding observations $\{\boldsymbol{x}_i\}_{i\geq 1}$.

Hidden Markov models allow the latent variable to be dependent on the (single) previous latent variable through a conditional distribution $p(\boldsymbol{z}_j|\boldsymbol{z}_{j-1})$, but it also assumes that it is independent of anything else. Formally, this is known as *Markovian* property:

$$p(\boldsymbol{z}_j|\boldsymbol{z}_{j-1}) = p(\boldsymbol{z}_j|\boldsymbol{z}_{j-1}, \dots, \boldsymbol{z}_1) \tag{2}$$

For observations, it is assumed that the $\boldsymbol{x}_j$ is only dependent on the corresponding latent variable $\boldsymbol{z}_j$ and, moreover, conditional on latent variable, the observation is independent of everything else.

Noting that the states must be discrete, we will divide the map into the grid of square cells, each one of them corresponding to a state. We will refer

to a state by a tuple (i, j), meaning the cell in the j-th row and i-th column (corresponding to x, y coordinates on the map).

The Markovian property (2) implies that it is natural to consider *transition* probabilities $p(\boldsymbol{z}_j|\boldsymbol{z}_{j-1})$ from one hidden state to the other. The transition probabilities model the usual movement of the bee without taking any observations into account. For example, if the bee is in the hive, it is more likely that after a second it will still be in the beehive than that it will suddenly appear kilometres away from it.

The transition probabilities can be concisely expressed as a transition matrix $\mathbf{A}$ such that

$$\mathbf{A}_{ij} = p(\boldsymbol{z}_t = j|\boldsymbol{z}_{t-1} = i) \tag{3}$$

The initial probabilities of the model starting in the i-th state (bee being in a location denoted by i-th state) are given by $\boldsymbol{\pi}_i = p(\boldsymbol{z}_1 = i)$.

To fully specify the model, we also need to define the probabilities of observations given that the bee is in any particular state. The conditional probabilities $p(\boldsymbol{x}_t|\boldsymbol{z}_t)$ are modelled as Gaussian distribution of the difference of presumed and real signal strength. More precisely, each state is naturally associated with location on a map (for consistency, we treat the location of the state as the location of the middle point of the cell). Given some potential location of a bee and a radio beacon location, we can calculate the ideal, noiseless signal strength which would be observed by the ground station. We look at the difference between the observed and ideal signal strengths and use a Gaussian distribution on this error. Technically, if the bee is at state a which is a cell with centre at (x_{bee}, y_{bee}) and we consider an observation from the radio beacon at location (x_{obs}, y_{obs}) with signal strength s_{obs}, then we model the probability by

$$\begin{aligned}\mathbb{P}(\boldsymbol{x}_t = (x_{obs}, y_{obs}, s_{obs})|\boldsymbol{z}_t = a) \\ = \mathcal{N}(s_{obs} - \sqrt{(x_{obs} - x_{bee})^2 + (y_{obs} - y_{bee})^2}|0, \sigma^2)\end{aligned} \tag{4}$$

Our goal will be to find the most likely *path* of the bee (sequence of hidden states), given the sequence of observations, that is, the maximum a posteriori (MAP) estimate

$$\arg\max_{\boldsymbol{z}_{1:T}} p(\boldsymbol{z}_{1:T}|\boldsymbol{x}_{1:T}) \tag{5}$$

Note that this is very different from calculating the sequence of most likely states for each observation, as those may not even form a valid path. For efficient calculation of the most likely *path*, we use the Viterbi algorithm, presented in the next section.

2.2 The Viterbi algorithm

The problem of finding the most likely sequence of latent states is efficiently solved by using the Viterbi algorithm (Forney 1973).

In the essence of this algorithm there is a dynamic programming approach: we will dynamically compute the probability of most likely path up to timestep t ending in state j, denoted by $\delta_t(j)$.

This computation will make use of the observation that this problem exhibits an *optimal substructure property*: that is, an optimal solution can be efficiently computed from the optimal solutions to the smaller subproblems. In this case, this means that the most likely path up to timestep t ending in j will consist of the most likely path up to timestep $t-1$ ending in i and then a transition from i to j, for some state i.

Formally, let us denote the probability of getting the current observations if we are in j-th state by $\phi_t(j) = p(\boldsymbol{x}_t|\boldsymbol{z}_t = j)$. Recall that a probability of a transition from state i to j is denoted A_{ij}. Then we can write the probability of most likely path up to time t ending in state j as:

$$\delta_t(j) = \phi_t(j)(\max_i \delta_{t-1}(i)A_{ij}) \tag{6}$$

As values of A_{ij} are known (after the training stage), and $\phi_t(j)$, the observation probabilities, are easy to compute, the problem is efficiently solvable using dynamic programming, by memorising the values of $\delta_t(j)$.

Also note that adding a new datapoint at time $T+1$ is easy: we only have to calculate $\delta_{T+1}(j)$ for all values of j, and we can make use of previously calculated δ_T.

2.3 Gaussian Processes

For the presentation of the framework of Gaussian processes we are largely going to follow Rasmussen and Williams (2005).

In the inference problem, we get some inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and some outputs $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ (in general, inputs and outputs are real-valued vectors). The key assumption is that for all input-output pairs, $\boldsymbol{y}_i = f(\boldsymbol{x}_i)$ for some unknown function f, possibly corrupted by noise.

Instead of assuming the general, parameterized form of f and trying to learn the parameters, as it is common in other machine learning approaches, Gaussian processes takes a different approach and models distributions over *functions*.

The distribution is fully characterised by the mean function $m(\boldsymbol{x})$ and the covariance function (also called *kernel*) $\kappa(\boldsymbol{x}, \boldsymbol{x}')$.

Let us choose some mean function $m(\boldsymbol{x})$ and the covariance function $\kappa(\boldsymbol{x}, \boldsymbol{x}')$. Then, for any finite set of inputs $\boldsymbol{t} = \langle t_1, \dots t_n \rangle$, the set of function values $f(\boldsymbol{t}) = \langle f(t_1), \dots f(t_n) \rangle$ has a jointly Gaussian distribution:

$$f(\boldsymbol{t}) \sim \mathcal{N}(\mu(\boldsymbol{t}), K(\boldsymbol{t}, \boldsymbol{t})) \tag{7}$$

where $\mu(\boldsymbol{t})$ is a vector and $\boldsymbol{K}(\boldsymbol{t})$ is a matrix such that

$$\mu(\boldsymbol{t}) = \langle m(t_1), \dots m(t_n) \rangle \tag{8}$$

$$\boldsymbol{K}(\boldsymbol{p}, \boldsymbol{q})_{ij} = \kappa(\boldsymbol{p}_i, \boldsymbol{q}_j) \tag{9}$$

Now assume we know outputs $\boldsymbol{y}_1, \dots, \boldsymbol{y}_n$ at inputs $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$. In addition, we want to make predictions at inputs $\boldsymbol{x}_1^*, \dots, \boldsymbol{x}_n^*$. Let us denote the predicted values by $\boldsymbol{y}_1^*, \dots, \boldsymbol{y}_n^*$.

Now, by the definition of Gaussian processes,

$$\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{y}^* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu(\boldsymbol{x}) \\ \mu(\boldsymbol{x}^*) \end{pmatrix}, \begin{pmatrix} K(\boldsymbol{x}, \boldsymbol{x}) & K(\boldsymbol{x}, \boldsymbol{x}^*) \\ K(\boldsymbol{x}^*, \boldsymbol{x}) & K(\boldsymbol{x}^*, \boldsymbol{x}^*) \end{pmatrix} \right) \tag{10}$$

By the well-known rules of conditioning multivariate Gaussians (see Murphy 2012, sec. 4.3) the posterior distribution is given as

$$y^* | y, x, x^* \sim \mathcal{N}(\hat{\mu}, \Sigma) \tag{11}$$

$$\hat{\mu} = \mu(\boldsymbol{x}^*) + \boldsymbol{K}(\boldsymbol{x}^*, \boldsymbol{x}) \left(\boldsymbol{K}(\boldsymbol{x}^*, \boldsymbol{x})\right)^{-1} (\boldsymbol{y} - \mu(\boldsymbol{x})) \tag{12}$$

$$\Sigma = \boldsymbol{K}(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{K}(\boldsymbol{x}^*, \boldsymbol{x}) \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x})^{-1} \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}^*) \tag{13}$$

Note that this calculation explains one of the strengths of Gaussian processes – it does not simply provide a prediction as a point estimate, but rather gives a Gaussian distribution centred around the best guess with well-calibrated variance, expressing uncertainty about the prediction.

As noted by Murphy (2012), a common practice is to set $m(\boldsymbol{t}) = \boldsymbol{0}$, since the Gaussian processes are flexible enough to approximate the function through covariance matrix alone.

However, so far, we have said nothing about the choice of covariance function κ. It is chosen to reflect the prior beliefs about the structure of function – how its output varies with changing input.

As noted by Genton (2002), the commonly used kernels in time series and spatial tracking are *stationary*, that is, the output of a kernel belongs only on the distance between the input vectors. This naturally embodies the idea that the absolute location or time is not relevant, only the relative distance between the datapoints and delay between them.

In this report, inspired by Mann et al. (2009) we consider only the Matérn 3/2 kernel, belonging to the class of stationary kernels. It is given by

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = M\left(||\boldsymbol{x} - \boldsymbol{x'}||\right) \tag{14}$$

where

$$M(d) = \delta^2 \left(1 + \frac{\sqrt{3}d}{\rho}\right) \exp\left(-\frac{\sqrt{3}d}{\rho}\right) \tag{15}$$

for the hyperparameters δ and ρ. Hyperparameter δ adjusts the scale of variations over the function values (output scale), while ρ determines the distance over which function values become uncorrelated.

2.4 Application to the problem

The training data consists of datapoints: a 5-tuple $\langle x_{bee}, y_{bee}, x_{radar}, y_{radar}, s\rangle$ for each timestep, where x_{bee}, y_{bee} are the x and y coordinates of a true bee location, respectively, and x_{radar}, y_{radar} are the x and y coordinates of a radar which made the observation and s is the signal strength.

We will formalise this as a *multi-output* function with input being time:

$$f(t) = \langle x_{bee}, y_{bee}, x_{radar}, y_{radar}, s\rangle \tag{16}$$

Following the suggestion by Osborne (2010, p. 51), we can treat this as a function g taking a timestep and a discrete label and outputting only scalars:

$$g(t, 0) = x_{bee} \tag{17}$$
$$g(t, 1) = y_{bee} \tag{18}$$
$$g(t, 2) = x_{radar} \tag{19}$$
$$g(t, 3) = y_{radar} \tag{20}$$
$$g(t, 4) = s \tag{21}$$

As noted by Osborne (2010, p. 51), this gives us the advantage that we can readily deal with partial and missing data. In training, we will learn the correlation between true location of a bee and radar data, and in the test time, we will input the radar observations and will ask to predict the bee location.

Furthermore, the missing data is gracefully handled – if the bee is out of range for all the radars in a particular time step, we simply do not input the radar location and signal strength for this timestep.

Note that if we have chosen an alternative formalisation of the prediction, that is, to feed in the datapoint along the timestep as an input:

$$f(t, x_{radar}, y_{radar}, s) = \langle x_{bee}, y_{bee}\rangle \tag{22}$$

we would be unable to treat missing data: it would be impossible to predict the bee location for the timestep with missing data, and we could not predict the bee location in the future.

2.5 The linear model of coregionalisation

It remains to specify the kernel for our formalisation which would take into the account that the second input to function f is a discrete label.

Following Álvarez, Rosasco, and Lawrence (2012, p. 12), we consider a type of multi-output kernel: sum of separable kernels. That is, the output will be a weighted sum of kernels of the input without a label:

$$\mathbf{K}([x,l],[x',l']) = \sum_{q=1}^{Q} k_q(x,x')\hat{k}_q(l,l') \tag{23}$$

or, equivalently,

$$\mathbf{K}([x,l],[x',l']) = \sum_{q=1}^{Q} k_q(x,x')(\mathbf{B_q})_{l,l'} \tag{24}$$

where each $\mathbf{B_q}$ is some 5×5 matrix (as the output is a 5-tuple, thus we have 5 possible labels).

We can consider the form of our function under this kernel. It is not hard to see that the outputs are expressed as a weighted combination of functions only depending on the timestep:

$$g(t,l) = \sum_{q=1}^{Q} w_{q,l} u_q(t) \tag{25}$$

The functions $u_q(t)$ are called *latent*. Furthermore, they are independent, have zero mean and covariance

$$\mathrm{Cov}[u_q(t), u_{q'}(t')] = \begin{cases} k_q(t,t') & \text{if } q = q' \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

This model is simple enough to be computationally tractable, yet powerful enough to learn the correlations between different labels.

2.6 Bayesian Cramér-Rao bounds

Our presentation of the Bayesian Cramér-Rao bounds is largely going to follow Dauwels (2005).

Let us consider the problem of estimating some states from the observations. Let $X = (X_1, X_2, \ldots, X_n)^T$ be the vector of states and let $Y = (Y_1, Y_2, \ldots, Y_N)^T$ be the vector of observations, where all X_k and Y_k are real-valued vectors, not necessarily of the same dimensionality. Furthermore, let $p(x, y)$ be a joint probability density function of X and Y.

Consider the estimation of X from the observations Y, that is, we have the estimator of X denoted by $\hat{x}(Y)$. We are interested in the error of this estimator and to this end we define the *error matrix* of this estimator:

$$\mathbf{E} := \mathbb{E}_{XY}[(\hat{x}(Y) - X)(\hat{x}(Y) - X)^T] \tag{27}$$

The error matrix relates to the mean square error, which is simply a weighted sum of diagonal elements:

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{x}(Y)_i - X_i)^2 = \sum_{i=1}^{n}\frac{1}{n}\mathbf{E}_{ii} \tag{28}$$

As we shall see, the error matrix is related to the *Bayesian information matrix* $\mathbf{J}$ defined as:

$$\mathbf{J}_{ij} := \mathbb{E}[\nabla_{x_i} \log p(x, y) \nabla_{x_j}^T \log p(x, y)] \tag{29}$$

where ∇ refers to the gradient, which, for vector $v = (v_1, \ldots v_q)^T$, is defined as

$$\nabla_v := \left[\frac{\partial}{\partial v_1}, \frac{\partial}{\partial v_2}, \ldots, \frac{\partial}{\partial v_q}\right]^T \tag{30}$$

Van Trees (1968, p. 72-73) proved a bound on the error matrix (27):

$$\mathbf{E} \succeq \mathbf{J}^{-1} \tag{31}$$

where inequality means that matrix $\mathbf{E} - \mathbf{J}^{-1}$ is positive semi-definite.

The theorem is subject to some regularity conditions and "weak unbiasedness" condition. However, Dauwels (2005) proved that if the latter condition does not hold, we still get the lower bound, however, not as tight. This bound by Van Trees is also known as "Bayesian Cramér-Rao bound" ("Bayesian CRB") or "posterior CRB" or "Van Trees bound".

Note that if we can calculate the Bayesian information matrix $\mathbf{J}$, it is particularly easy to bound the mean square error, which can be thought as a performance measure of our estimator. By the bound (31) and the fact that diagonal submatrices of a positive semi-definite block matrix are themselves positive, if follows that

$$\sum_{i=1}^{n}\frac{1}{n}\mathbf{E}_{ii} \succeq \sum_{i=1}^{n}\frac{1}{n}[\mathbf{J}^{-1}]_{ii} \tag{32}$$

and by equation (28), this is a bound of mean square error.

Therefore, to get the bounds on the performance of our predictor, we only need to calculate the Bayesian information matrix $\mathbf{J}$, which is a $n \times n$ matrix for n observations. However, the computation of the elements of $\mathbf{J}$ is computationally expensive and so is the inversion of it. To make the computation tractable, we are going to use the result of Dauwels (2005) which allows to take into the account the structure of $p(x, y)$ and compute the Bayesian information matrix more efficiently.

To this end, we consider the state-space model with freely evolving state, that is, we say that the next state X_{k+1} depends only on the state X_k, and the observations Y_k depend only on the corresponding state X_k. The probability density function of such model is

$$p(x, y) = p_0(x_0) \prod_{k=1}^{n} p(x_k|x_{k-1}) p(y_k|x_k) \tag{33}$$

To derive the results for filtering, we can use the results of Tichavsky, Muravchik, and Nehorai (1998). For the derivation, the interested reader is referred to Dauwels (2005) and Tichavsky, Muravchik, and Nehorai (1998), but for our purposes it is enough to state this (rather technical) result: The Bayesian Cramér-Rao bound for filtering is

$$\mathbf{E}_{kk} \succeq (\boldsymbol{\mu}_k^F)^{-1} \tag{34}$$

where $\mathbf{E}$ is the Bayesian information matrix (29), and $\boldsymbol{\mu}_k^F$ is defined recursively as

$$\tilde{\boldsymbol{\mu}}_{k+1}^F = \boldsymbol{G}_{k,22} - \boldsymbol{G}_{k,21}(\boldsymbol{\mu}_k^F + \boldsymbol{G}_{k,11})^{-1}\boldsymbol{G}_{k,12} \tag{35}$$

$$\boldsymbol{\mu}_{k+1}^F = \tilde{\boldsymbol{\mu}}_{k+1}^F + \boldsymbol{\mu}_{k+1}^Y \tag{36}$$

with the matrices $\boldsymbol{G}$ being defined as

$$\boldsymbol{G}_{k,11} = \mathbb{E}[\nabla_{x_k} \log p(x_{k+1}|x_k) \nabla_{x_k}^T \log p(x_{k+1}|x_k)] \tag{37}$$

$$\boldsymbol{G}_{k,12} = \mathbb{E}[\nabla_{x_k} \log p(x_{k+1}|x_k) \nabla_{x_{k+1}}^T \log p(x_{k+1}|x_k)] \tag{38}$$

$$\boldsymbol{G}_{k,21} = [\boldsymbol{G}_{k,12}]^T \tag{39}$$

$$\boldsymbol{G}_{k,22} = \mathbb{E}[\nabla_{x_{k+1}} \log p(x_{k+1}|x_k) \nabla_{x_{k+1}}^T \log p(x_{k+1}|x_k)] \tag{40}$$

$$\boldsymbol{\mu}_k^Y = \mathbb{E}[\nabla_{x_k} \log p(y_k|x_k) \nabla_{x_k}^T \log p(y_k|x_k)] \tag{41}$$

and the recursion is initialised as

$$\tilde{\boldsymbol{\mu}}_0^F = \mathbb{E}[\nabla_{x_0} \log p(x_0) \nabla_{x_0}^T \log p(x_0)] \tag{42}$$

Analogously, we have a bound for smoothing, given by Dauwels (2005):

$$\mathbf{E}_{kk} \succeq (\tilde{\boldsymbol{\mu}}_k^F + \tilde{\boldsymbol{\mu}}_k^B + \boldsymbol{\mu}_k^Y)^{-1} \tag{43}$$

where

$$\tilde{\boldsymbol{\mu}}_k^B = \boldsymbol{G}_{k,22} - \boldsymbol{G}_{k,21}(\boldsymbol{\mu}_{k+1}^F + \boldsymbol{G}_{k,11})^{-1}\boldsymbol{G}_{k,12} \tag{44}$$

$$\boldsymbol{\mu}_k^B = \tilde{\boldsymbol{\mu}}_k^B + \boldsymbol{\mu}_k^Y \tag{45}$$

for matrices $\boldsymbol{G}$ defined in (38)–(41).

Note that while the formulas look cumbersome, the only difficulty in evaluating the equations is the necessity to calculate the gradients of function of pdfs and also to be able to take expectations over this expression. We will see that as we can easily generate sequences of x_k and y_k, calculating the expectations is relatively easy by Monte Carlo methods. However, calculating gradients of logarithm of pdfs require an explicit representation of the density and we will deal with it with methods presented in Section 2.7.

2.7 Kernel density estimation

The bounds on the performance described in Section 2.6 requires the value of probability density function $p(x_{k+1}|x_k)$ and $p(y|x)$, which also needs to be differentiable. However, the simulation of the beehive is far too complex to provide any analytic solution for the pdf, thus the only thing we can do is to generate sequences of bee state (x_k) and observations (y_k), that is, sample from the joint distribution.

We employ the technique known as *kernel density estimation* to provide an estimate of the density function. For introduction, see Simonoff (1998, p. 112). The basic idea is to sample many points from the distribution and calculate the value of a pdf depending on how close it is to the points in sample.

Specifically, if we sample real-valued vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ from the distribution, then the estimate of probability density function is

$$\hat{p}(\boldsymbol{x}) = \frac{1}{n|H|} \sum_{i=1}^{n} K_d(H^{-1}(\boldsymbol{x} - \boldsymbol{x}_i)) \tag{46}$$

where

- K_d is the kernel function, providing some measure of similarity between the data points. We will use

$$K_d(\boldsymbol{u}) = \prod_{i=1}^{d} \varphi(u_i) \tag{47}$$

for φ being the pdf of a univariate standard normal distribution. In case datapoints are scalars, the kernel density estimate becomes a simple mixture of Gaussians centred around the datapoints.

- H is the positive definite, symmetric matrix, called *bandwidth* matrix. It changes the scale – that is, it tells how close the datapoints should be in order for them to have significant effect on the result.

  It is helpful to think of the case in a single dimension – then the summands are $K_d\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right)$, and the bandwidth parameter naturally controls the scale.

As noted, for the kernel we are going to use a simple product of standard normal Gaussians. The choice of the kernel is arbitrary, but our choice is motivated by the simplicity of this distribution and also the convention that Gaussian is a preferred distribution in machine learning due to it being a maximum entropy distribution with a given mean and variance.

Having made this choice, we can rewrite the estimate as a mixture of multivariate Gaussians:

$$\hat{p}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n}(2\pi)^{-\frac{d}{2}}|H|^{-\frac{1}{2}}\exp\left(-\frac{1}{2}(x-x_i)^T H^{-1}(x-x_i)\right) \tag{48}$$

For this to be used as an estimator, we just need to choose a matrix H, which will be discussed in the Section 6 on applying this method for our project.

3 Modelling

In order to evaluate the tracking approaches, we need to acquire the training and testing data and apply the general theory of both Gaussian Processes and Hidden Markov models to our specific problem.

3.1 Hive simulator

In order to use machine learning and evaluate the results, it is required to have large amounts of sample bee flight data. Unfortunately, the real bee flight data was not available at the time of the project development, thus a bee simulator was used instead. This has an advantage of readily generating new paths for training and testing, thus eliminating the need to maintain careful separation of test, training and validation datasets.

The bee flight simulator is taken from the beehive simulation developed by Williams and De Souza Junior (2016) in Python with minor modifications. It

aims to model realistic flight paths by using extensive background knowledge on the bees' orientation in space.

The model simulates the habitat of bees in the South Eck catchment region in north-eastern Tasmania. The space is discretised into spatial grid consisting of 10×10 m grid cells and the simulation is implemented as a discrete-time process, simulating a bee location every 3 seconds.

The flight paths are generated by two types of random processes (Lévy or correlated random walks), as there is a debate over which approach is more suitable for modelling animal foraging paths (Kawai and Petrovskii 2012). We choose a correlated random walk model.

The simulator provides an extensive simulation of a beehive, including passing the information about locations of the food sources by waggle dance and different bee roles, thus producing realistic, unlimited training and testing data. However, it also contributes to the limitations of the model, as the maximum time granularity cannot be more that 3 seconds, the location precision is limited to 10 meters and it depends on the assumption that bee flight paths can be modelled as random walk of particular type.

3.2 Sensors simulation

To complete the simulation, we also need to simulate the ground sensor observations. We expect these to be corrupted by noise in the real life, so we add independent and identically distributed (iid) Gaussian noise to our idealised signal strength samples.

We place 100 sensors in the equally-spaced grid formation in the 1.9 km $\times$1.9 km simulated field with the hive in the centre (this slight asymmetry is chosen so that the hive, where a significant amount of time is spent, is in range for some radar). The presumed range of the radar is simulated to be a circle with radius of 100 meters. This decision is influenced by requirement (R2): the number of sensors is limited and thus they are placed quite sparsely.

The procedure for generating an observation for a particular ground station given the bee location is as follows: we determine the distance between the bee and the station, calculate the signal strength, corrupt it by noise and check if it is below a certain threshold – if so, we record an observation with noisy signal strength, otherwise we say that no detection occurred.

The signal strength for each sensor is calculated following the simple free-space model (Fujimoto, Riley, and Perumalla 2006, p. 31). Given its distance to the bee d, the signal strength is:

$$-n10\log_{10}\left(\frac{d}{d_0}\right) + S_{dB} \tag{49}$$

where:

- n is a path-loss exponent. It is chosen to be $n = 2$, same as vacuum path-loss exponent, as hive is assumed to be located in the open field, with no significant objects to obstruct the signal.

- d_0 is a reference distance (signal strength is calculated relative to strength at this distance). A typical value, also chosen in this simulation, is $d_0 = 1$ m.

- S_{dB} is a zero-mean Gaussian (normal) random variable, $S_{dB} \sim \mathcal{N}(0, \delta^2)$. The typical value for standard deviation δ is in range from 3 to 12, depending on the perceived noisiness of the environment (Fujimoto, Riley, and Perumalla 2006, p. 31).

  We conservatively choose a value of $\delta = 8$, for a pessimistic view on how noisy the sensors in an open field might be to account for various real-life inaccuracies not reflected in a model.

In the generated data, each testing datapoint is of two types: either it consists of a location of a ground sensor which detected the bee accompanied by the signal strength or it simply notes that the bee was not detected by any radar (*censored observation*). Training datapoints in addition have a true bee location.

The simulation is done in Python, and documentation generation tool `Sphinx` is used to provide both code documentation and a high-level description of the code.

3.3 Gaussian processes

We use the theory described in Sections 2.3, 2.4 and 2.5 to model the bee flight using a Gaussian process.

In the linear model of coregionalisation, we use three latent processes, that is, $Q = 3$ in equation (25), meaning that predictions are a weighted sum of three functions. As noted before, for the latent functions we are using Matérn 3/2 kernels. The parameters are optimised using maximum likelihood.

We implement the Gaussian processes prediction framework using `GPy` library in Python, as it provides the framework for maximum likelihood optimisation and predictions.

3.3.1 Optimisations

Given that our first task is to compare different machine learning approaches based on accuracy, for now we are not too concerned with the prediction speed or memory usage as long as the testing can be done in some reasonable time.

However, it is important to consider potential optimisations which would be implemented in practice to consider their effect on accuracy and the design of the framework.

One such concern is the different modes of prediction. For Gaussian processes, smoothing (offline prediction with full knowledge of the data) is the natural mode of operation, as all available data is simply added to the model with appropriate timestamps, and subsequently we ask for predictions at particular points.

However, filtering is not straightforward: we must deal with datapoints coming one-by-one. Note that conditioning on more datapoints implies calculation of an inverse of a matrix with all observations so far, as evident from equation (13). For testing the accuracy, the simplest thing to do is, upon the arrival of a new datapoint, to throw away all the information and recalculate all parameters from scratch. This does not influence accuracy and is easy to implement given the existing functions in `GPy` library.

However, we note that Osborne (2010, p. 83) provides a way to reuse the computation of the Gaussian Process covariance matrix by using a Cholesky decomposition. This would remove the bottleneck present in the current implementation without any changes to the accuracy.

Another possible optimisation would be to model x and y coordinates using two separate Gaussian processes. This would reduce the memory usage during training stage, as both coordinates can be trained separately, thus allowing to use more data, hopefully yielding better predictions.

More precisely, instead of formalising a Gaussian process as trying to predict the function outputting a 5-tuple (as described in Section 2.4), we consider a function with time as input:

$$f_x(t) = \langle x_{bee}, x_{radar}, s \rangle \tag{50}$$

where s is, as before, signal strength, and x_{bee}, x_{radar} are bee true location and radar x-coordinates, respectively.

However, this might influence the accuracy, thus we want to test if the optimisation is worth the effort. We compare the approach of having two separate Gaussian processes for x and y coordinates with having a single GP for both coordinates. We train both models on same 30 minutes of bee flight, and test them on another 30 minutes of bee flight in both smoothing and filtering modes.

	Filtering RMSE (m)		Smoothing RMSE (m)	
Frequency	Separate GPs	Single GP	Separate GPs	Single GP
3 s	180.5	118.5	124.5	71.5
6 s	195.0	125.9	129.6	80.4
9 s	233.6	168.9	150.7	105.7
12 s	228.3	144.2	146.6	92.3
15 s	253.2	188.0	170.6	136.9
18 s	273.9	189.2	183.6	136.7
21 s	302.1	262.6	187.0	182.2
24 s	251.7	165.0	164.4	116.0

Table 1: Root mean square error of bee location predictions, either in filtering or smoothing modes. Separate GPs refer to the approach where x-coordinate and y-coordinate are modelled by two separate Gaussian processes, and Single GP refers to the situation where the location is modelled as a single Gaussian process. Frequency is the delay between subsequent observations.

The results can be seen in Table 1. Note that using separate GPs causes a significant drop in accuracy. We conclude that it is better to model the bee flight as a single function and we will use this approach for the rest of the report.

3.3.2 Hyperparameters: Matrix size of the Gaussian processes

Another possible optimisation is limiting the number of considered past observations.

In the filtering problem (where datapoints come one-by-one), we only care about the prediction of the most recent location of a bee. Since we are using Matérn kernels, the prediction depends strongly on the recent observations and the effect of observations far in the past is small. This is because Matérn kernel value depends only on the distance of two datapoints (which, in our case, in time difference) and this dependence is controlled by the *lengthscale* parameter ρ.

$$K(d) = \delta^2 \left(1 + \frac{\sqrt{3}d}{\rho}\right) \exp\left(-\frac{\sqrt{3}d}{\rho}\right) \tag{51}$$

In the coregionalised model of Gaussian process we are using 3 Matérn kernels, and the largest value of lengthscale parameter of ρ, estimated by maximum likelihood, is 68 three-second intervals, corresponding to about 3 minutes of flight. Therefore, we conjecture that if we only input roughly the

last 70 observations instead of full history, the decrease in accuracy should be negligible.

Note that this is important for speed: as predictions requires inverting the kernel matrix of the observations, reducing the number of most recent points which the model takes into the account decreases the running time significantly. Therefore, we seek to ascertain the effect of the number of previous observations and test our conjecture empirically.

The results are outlined in Table 2. They suggest setting the history cutoff value to about 70, as setting anything above makes no significant difference on the accuracy while compromising the running time.

Most recent obs.	RMSE mean (m)	RMSE SD (m)	Time (s)
5	166.32	4.24	21.88
10	131.89	4.73	24.56
20	128.52	5.41	27.16
30	127.34	5.07	33.55
40	127.20	4.65	35.10
50	127.30	4.57	40.73
60	127.29	4.59	45.71
70	127.26	4.60	54.07
80	127.25	4.61	58.46
90	127.25	4.61	70.06
100	127.25	4.61	81.36
200	127.25	4.61	167.29

Table 2: Comparison of the root mean square error depending of the history size. The SD stands for standard deviation, as the results are obtained by averaging over 10 samples, each predicting 30 minutes of bee flight. The time denotes the average time taken per each sample.

We have developed the Gaussian process framework for prediction and selected the relevant optimisations which reduce the prediction time and memory costs without significantly affecting the accuracy. Now we turn our attention to the Hidden Markov model approach, and compare the performance of both approaches in Section 4.

3.4 Hidden Markov model

We implement the Hidden Markov models following the basic definitions in Section 2.1.

The algorithm is easy enough to be implemented from scratch, so we choose not to use any specific libraries for hidden Markov models, giving us

the freedom to optimise all parts of the process, easily employ multithreading and optimise for our particular setting.

The project is mostly implemented in Python, which is generally slower than compiled languages. Thus, for HMMs, we use C++, which allows us to compile the code for a particular architecture. In addition, we gain a fine-grained control over the concurrency management. Both of these benefits contribute to significant speed gains.

3.4.1 Optimisation: convolutional approach

To improve the model, we will perform some domain-specific simplifications. The method to estimate the transition matrix directly is well-known, described, for example, by Ghahramani (2002). While the second optimisation – to limit the considered states in the inference step – is simple and was used, for instance, by Shih, Renuka, and Rose (2015), our contributions include applying this technique to the spatial domain by using domain-knowledge, thus enabling the efficient implementation of the first simplification.

First we note that, unlike in a general setting, we can directly observe the latent state during training, as we know the true bee location. Therefore, we can estimate the parameters of transition matrix $\mathbf{A}$ (defined in equation (3)) directly by checking the number of empirical transitions between the states.

Furthermore, we employ a convolutional approach: as the hidden states represent a discretised grid (with natural order), and the top speed of a bee is limited, we know that transitions are only possible to "neighbour" states – the bee cannot transition from any state to any other, as for example, it is unrealistic to model a bee flying through the whole map in a single time step.

Implementing the original version of hidden Markov model would require calculating $O(N^2)$ operations for each timestep, where N is the number of states in the model. Since the number of states depends quadratically on the cells per x or y coordinate, the path estimation (Viterbi algorithm) becomes prohibitively expensive. Moreover, as we will infer $\mathbf{A}$ using the sample flights, estimating the transition probabilities of the states far away from the centre would require a large amount of data, as flights are naturally centred around the hive.

The *convolutional* approach alleviates both of these problems. In this approach, we assume that the transition probability depends only on the difference between locations, not on the absolute position on the map. More precisely, we will assume that

$$p(\boldsymbol{z}_t = (x + \Delta x, y + \Delta y) | \boldsymbol{z}_{t-1} = (x, y)) \tag{52}$$

is the same for all x, y. Now estimation becomes easy, as we simply generate n datapoints for bee flight path $(x_1, y_1), \ldots (x_n, y_n)$ and estimate the transition probabilities by a proportion of such transitions in the training data:

$$p(\boldsymbol{z}_t = (x+\Delta x, y+\Delta y)|\boldsymbol{z}_{t-1} = (x, y)) = \frac{1}{n}\sum_{i=2}^{n} \mathbb{I}(x_i - x_{i-1} = \Delta x \wedge y_i - y_{i-1} = \Delta y) \tag{53}$$

This helps us in two ways: firstly, by inspecting the transition matrix, we can find out that transitions are possible to only a few neighbouring cells – this will dramatically decrease the running speed, as now it is only $O(Nk)$ per each timestep (where k is the number of states it is possible to transition in a single step).

Furthermore, the estimation of the transition matrix becomes feasible, as number of estimation parameter decreases from $O(N^2)$ to $O(k)$ and we no longer have a problem of having few observations for distant parts of the map.

Combined together, these optimisations provide considerable reductions in running time and memory usage and mitigate the sparsity problem, making learning feasible.

3.4.2 Hyperparameters: grid size

In order to fully specify the Hidden Markov model, we must determine the values of hyperparameters. In this case, there is only one: the size of map grid. In discretisation, we have to choose the size of cells – as they are square, it is enough to choose the length of its side.

This has dramatic implications for the running time: denoting cell side length as δ, each timestep takes $O(Nk)$ time, where both the number of states $N \propto 1/\delta^2$ and neighbouring cells $k \propto 1/\delta^2$ depend on δ. This shows that running time has $1/\delta^4$ dependency, thus even small increase in δ allows to significantly reduce the running time.

To ascertain if we can do so without losing accuracy, we employ a grid search over δ, with a sample size of 30 min of honey bee flight (with datapoints every 3 seconds), averaged over 50 such samples.

Results of the simulation are shown, in detail, in Table 3 and plotted (without running time) in Figure 1. The data implies that the accuracy is almost identical for small values of cell size (10–20 m), while the running time drops dramatically while choosing the slightly larger value. Motivated by this, we will generally set the value of $\delta = 20$ m in subsequent exploration of model behaviour.

Cell size (m)	RMSE mean (m)	RMSE SD (m)	Time (s)
5	64.31	13.16	85.5
10	56.84	9.69	9.6
15	56.18	8.47	3.3
20	54.06	5.56	1.8
25	56.42	5.19	1.2
30	58.18	8.72	0.9
35	61.09	11.24	0.7
40	62.35	5.62	0.6
45	62.30	6.82	0.5
50	68.84	7.04	0.5
55	62.97	10.76	0.5
60	62.78	8.49	0.4
65	73.16	9.93	0.4
70	50.08	13.91	0.4
75	51.59	14.35	0.4
80	83.77	13.12	0.4
85	72.58	5.27	0.4
90	61.55	7.33	0.4
95	74.10	17.87	0.4
100	105.25	5.66	0.4

Table 3: Accuracy (root mean square) depending on cell size of the discretisation grid. SD stands for standard deviation, time marks the total time for predictions with the given cell size.

3.4.3 Censored observations

As the observations come at precise and known times, we still get some information even if no ground radar detected the bee – specifically, bee is more likely to be in areas not covered by any radar. Such a missing but informative observation is called a *censored* observation.

Note that it is difficult to incorporate information from censored observations to Gaussian processes framework. However, for Hidden Markov models this requires only a slight change of likelihood calculations.

To include the information from the censored observations in our model, for all cells on the discretised map we must compute the likelihood of bee being in that location and avoiding detection from all ground sensors. Since we have assumed that the noise is independent for all the sensors, we can calculate the likelihood by calculating it separately for all ground radars and

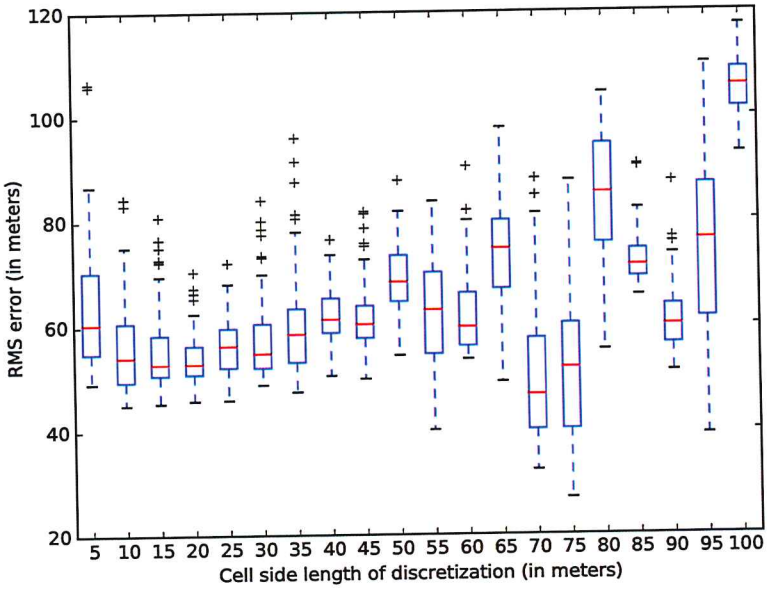

Figure 1: Box plot of root mean square error in meters, depending on the chosen cell side length. Whiskers are of 1.5 interquantile length, with datapoints beyond treated as outliers.

multiplying it together:

$$\mathbb{P}(\{\text{bee avoids detection}\}) = \prod_{i=1}^{n} \mathbb{P}(\{\text{bee avoids detection from } i\text{-th sensor}\}) \tag{54}$$

Given the bee and i-th ground sensor locations, we can use equation (49) to determine the signal strength s_i in a noiseless environment.

As our noise is Gaussian and additive,

$$\mathbb{P}(\{\text{bee avoids detection from } i\text{-th sensor}\}) = \mathbb{P}(s_i + \epsilon > r) \tag{55}$$

where r represents a maximal range of a ground radar and $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Knowing the range of the radar allows us to easily evaluate equation (55) and thus (54). Once we are able to compute the likelihoods for arbitrary locations, the rest of the algorithm remains the same. Further, calculation of these probabilities does not require any data and thus can be precomputed before the prediction stage to improve the running speed.

4 Comparison of the performance

To evaluate the performance of Hidden Markov models and Gaussian process approaches, we compare them on both filtering mode (where the data is coming datapoint-by-datapoint and bee's most recent location must be predicted) and smoothing mode (where the full flight path is predicted, given all observations at once).

For comparison, we also use two baselines: "Always-0" baseline predicts that the bee is always in the hive; "Radar baseline" takes the last 10 ground station locations from observations and averages them to generate a prediction.

By the requirement (R1) we aim to conserve power and thus send the radio beacon signal as infrequently as possible. To this end, we evaluate the performance of both models with observations coming at different frequencies (with different delays between consecutive observations). The results are averaged over 10 samples, with each sample consisting of 300 datapoints (for observations every 3 seconds, this is 15 minutes of flight, while for observations every 24 seconds, this corresponds to the flight duration of 2 hours). About 1.8% of datapoints are missing (censored). However, only HMMs use information from censored observations.

Frequency	HMM RMSE (m)	GP RMSE (m)	Always-0 baseline	Radar baseline
3 s	98.1 ± 15.1	126.3 ± 9.1	291.0 ± 198.1	203.2 ± 99.4
6 s	106.7 ± 19.4	155.7 ± 11.1	296.7 ± 128.6	258.7 ± 98.1
9 s	127.4 ± 27.4	176.2 ± 11.2	254.9 ± 119.4	231.3 ± 97.4
12 s	131.4 ± 30.5	194.5 ± 7.3	263.6 ± 95.9	249.8 ± 87.3
15 s	158.3 ± 31.0	202.7 ± 7.1	241.0 ± 86.6	230.5 ± 79.6
18 s	158.0 ± 31.8	209.8 ± 10.2	247.5 ± 76.8	237.1 ± 70.9
21 s	183.4 ± 27.6	223.3 ± 12.9	248.9 ± 69.6	240.1 ± 63.4
24 s	182.5 ± 29.1	233.4 ± 10.1	276.8 ± 49.6	265.6 ± 45.2

Table 4: Online prediction mode evaluation. The root mean square error (RMSE) is written as a mean, with standard deviation given after a ± sign. The baseline "Always-0" is root mean square error of always predicting that bee is at the hive. "Radar baseline" is the root mean square error of always predicting the average of 10 last radar locations.

The results for online prediction mode (for datapoints coming one-by-one) are detailed in Table 4. It seems that hidden Markov models outperforms the Gaussian process approach at all frequencies, also firmly beating the baselines.

The results for offline predictions are detailed in Table 5. This task is evidently easier, as the root mean square error is significantly lower than in the online prediction mode. The Hidden Markov model approach seems to be superior to the Gaussian processes approach also in this case.

Frequency	HMM RMSE (m)	GP RMSE (m)	Always-0 baseline	Radar baseline
3 s	63.3 ± 11.9	78.6 ± 8.7	291.0 ± 198.1	203.2 ± 99.4
6 s	64.9 ± 12.5	100.5 ± 11.1	296.7 ± 128.6	258.7 ± 98.1
9 s	83.8 ± 15.9	120.5 ± 10.6	254.9 ± 119.4	231.3 ± 97.4
12 s	94.4 ± 22.2	131.0 ± 8.1	263.6 ± 95.9	249.8 ± 87.3
15 s	114.4 ± 19.5	139.8 ± 6.5	241.0 ± 86.6	230.5 ± 79.6
18 s	114.6 ± 22.5	147.1 ± 14.2	247.5 ± 76.8	237.1 ± 70.9
21 s	136.7 ± 24.8	156.7 ± 12.0	248.9 ± 69.6	240.1 ± 63.4
24 s	137.7 ± 22.9	169.0 ± 11.5	276.8 ± 49.6	265.6 ± 45.2

Table 5: Offline prediction mode evaluation. The root mean square error (RMSE) is written as a mean, with standard deviation given after a ± sign. The baseline "Always-0" is the root mean square error of always predicting that bee is at the hive. "Radar baseline" is the root mean square error of always predicting the average of 10 last radar locations.

As Hidden Markov model approach outperforms the Gaussian processes in both offline and online predictions and presents no significant downsides except for inability to quantify uncertainty (provide variance estimate), we choose to use Hidden Markov models for our prototype bee tracking software implementation.

One hypothesis for the Hidden Markov models superiority is the discrete nature of observations. While the underlying bee trajectory is better modelled by continuous Gaussian Processes, the ground sensor locations are fixed and thus can be thought as discrete: this is possibly better modelled by HMMs. Also, HMMs allow specifying the precise relationship between observations and bee location, while Gaussian Processes must learn this from the data. In addition, only HMMs can gain information from censored observations. We speculate that the combination of these factors results in better performance for HMMs.

5 Implementation of tracking software

We have created a prototype bee tracker which is ready to be deployed in the field, features a intuitive visualisation of predicted bee paths and is lightweight enough to track hundreds of bees simultaneously.

The bee tracker is built on top of the existing system created by Prof. Alex Rogers. Our contributions include optimised prediction engine, visualisation of the predictions and communication between these parts via RedisDB and MongoDB.

5.1 Existing system

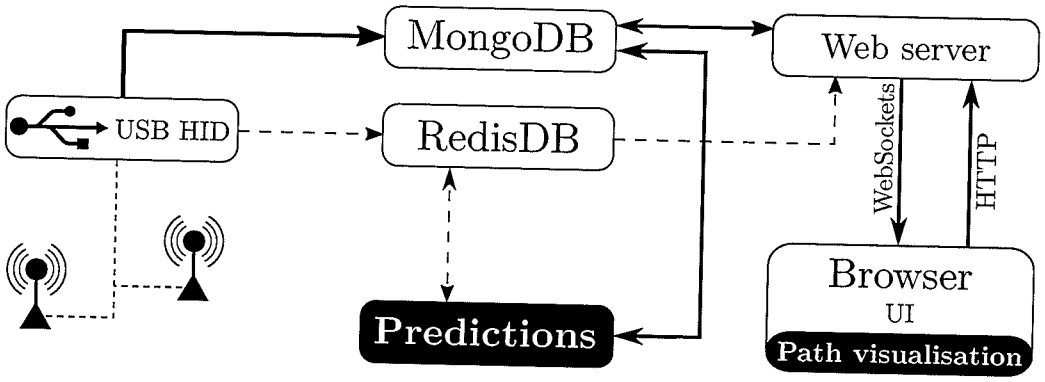

Figure 2: Diagram of bee tracker software components. Our contributions (marked in solid black) include a prediction module and changes to the UI running in browser to display a path prediction.

Prof. Alex Rogers has implemented the basic framework of the bee tracker software, consisting of the server for communication with the ground stations via USB, main web server in Node.js and website displaying the map and ground sensor updates using JavaScript (see Figure 2).

The system consists of several components, communicating via RedisDB messages and using MongoDB for persistent storage. The ground station data is retrieved via USB HID and written to the persistent MongoDB database, also generating the RedisDB publisher / subscriber message. This allows web server to wait for updates and send the messages indicating the arrival of new information via WebSockets to the browser. The user front end, implemented in JavaScript, retrieves the data via conventional HTTP requests. The user interface consists of a map indicating the location of deployed ground stations (rendered in Google Maps) and highlighting of stations which have just received an observation.

We will build on top of the existing framework, aiming to preserve modularity and thus communicating only via RedisDB and MongoDB databases. Furthermore, we will enhance the user interface to conveniently display the bee flight path.

5.2 Our contributions

5.2.1 Predictions module

We develop a module which, upon receiving a RedisDB message, retrieves the information about ground station observations, predicts the flight path, writes it to the MongoDB and informs the rest of the system via RedisDB message that new prediction paths are available.

The module is written in C++, following the considerations outlined in Hidden Markov models design (Section 3.4). Specifically, C++ offers the ability to produce highly optimised, natively compiled code yielding fast predictions. Also, C++ allows to use low-level concurrency primitives which let us devise a module running on several threads with minimal overheads.

The in-depth documentation of the code is provided using `Doxygen`. The documentation has been transformed into `Sphinx` format using `Breathe` plugin. This allows generating a single document for both the sample flight generator and predictions documentation.

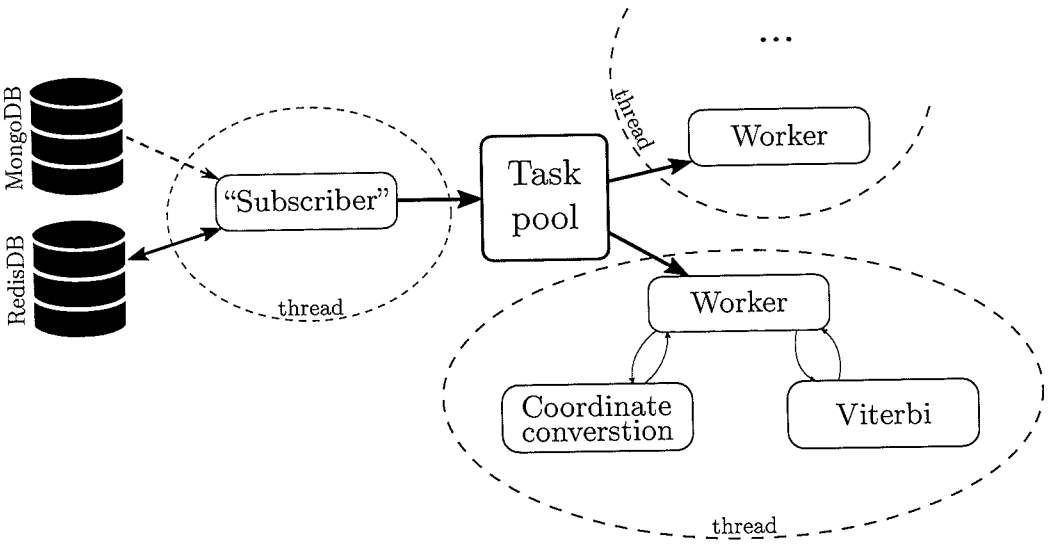

Figure 3: Abstract scheme of prediction engine implementation.

The communication is done through the RedisDB which supports publisher / subscriber mechanism. This avoids the need to check for new data periodically, instead allowing the threads to idle until a new message is published and all subscribers are informed of the message. The event driven approach offers the benefit of avoiding event loops which would waste resources idling and also allows processing with minimal latency.

Firstly, one distinct thread ("producer") subscribes itself to the RedisDB publisher / subscriber channel, thus receiving the messages about new observations (see Figure 3). If the thread receives a message about a new data point, it connects to the MongoDB database and retrieves unprocessed observations. Then it groups the updates by the bee which is being tracked and pushes new datapoints to the task pool.

Meanwhile, the worker threads are idling until they find a new task in the task pool. Upon receiving a request to update the specific bee with the given datapoints, the worker thread converts coordinates from latitude and longitude to coordinates relative to the placement of radars and pushes the observations to Viterbi algorithm class. Once it receives a prediction, it converts it back to latitude and longitude format and writes the newly predicted path to the MongoDB database. Finally, once all the worker threads have processed the information, the "producer" thread publishes a single RedisDB message to inform the rest of the system about the predicted paths.

The Viterbi algorithm class implements the algorithm developed in Section 3.4, using dynamic programming. We also use multithreading here, noting that the dynamic programming amounts to filling a table for each timestep, but each cell in this table depends only on the previous timesteps and thus can be computed concurrently. More precisely, as evident from equation 6, the values of $\delta_t(j)$ depends only on $\delta_{t-1}(i)$ for some indices i. We use a barrier as a concurrency management primitive, forcing threads to synchronise at each timestep. This ensures that previous timestep data is always computed whenever needed.

We use multithreading in two ways: if we have more bees than threads, it pays off to use the thread pool to predict different bees concurrently. On the other hand, if we have more threads than bees, it is beneficial to use parallelism in the path prediction for a single bee (with barriers), as otherwise some threads would be idling and thus wasting computing power.

As intended, this module follows the paradigm of storing persistent data (predicted paths) in MongoDB and sending status messages via RedisDB. This allows the module to be written in C++ while the rest of the system is not, as narrow interface ensures compatibility. The module provides highly optimised predictions engine acting on addition of new data with minimal latency.

5.2.2 User interface

We enhance the existing front end interface by plotting the predicted most likely path on the map for each bee.

The path is drawn on the map using Google Maps API for drawing polygons, allowing swift updates, as graphics are rendered in the browser.

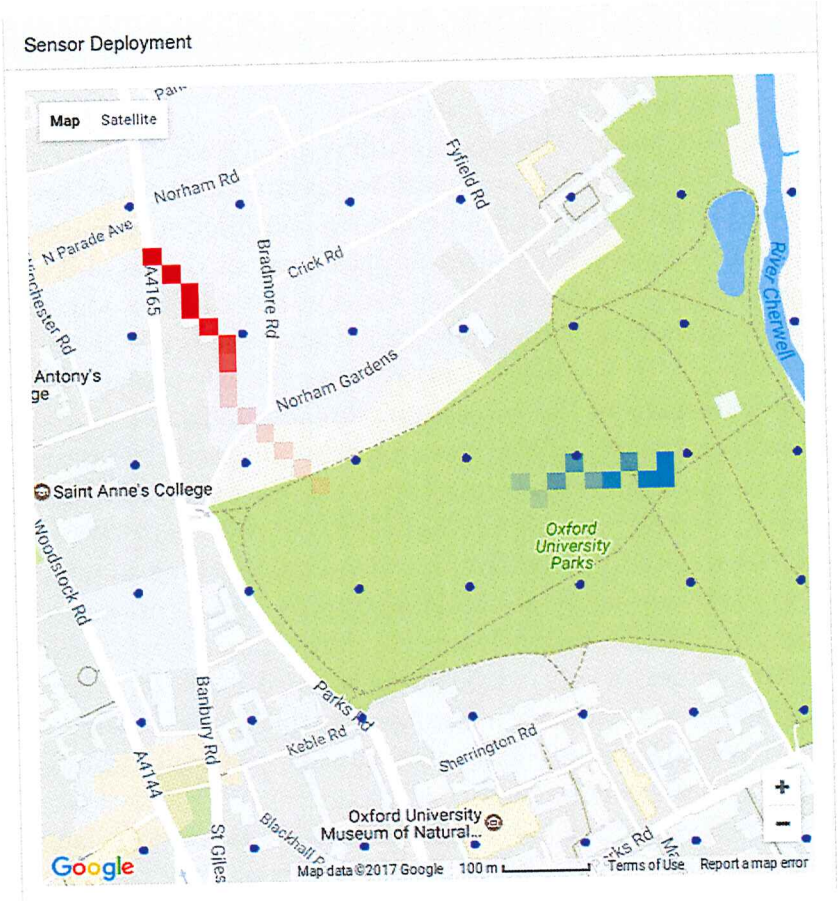

Figure 4: The user interface. The screenshot shows two bees, denoted by red and blue. Bright colours indicate recent observations, faded colours represent older ones.

The path is displayed either as continuous line or, to better visualise uncertainty, as fading blocks. We note that our prediction engine discretises the map into a grid, thus inherently cannot provide better estimates than the grid cell – we can visualise this uncertainty by highlighting the whole block. Recent predictions are represented by opaque blocks, the older predictions are visualised by faded colour, thus giving intuitive feeling of time and uncertainty. Path of different bees are displayed using different colours.

The screenshot of a user interface can be seen in Figure 4.

5.3 Optimisations

Our goal is to support the tracking of the bees for an unlimited amount of time. However, in the naïve version of Viterbi algorithm, we need to keep a dynamic programming table (consisting of probabilities for every cell in discretised map grid) for each timestep. This means that the memory requirements are growing linearly with each added observation.

To alleviate this problem, we observe that adding a new observation has a tiny effect for predictions far in the past. Therefore, we will keep dynamic tables only for some recent observations and regard the distant past as fixed, meaning that we will only keep the most likely path for old data.

To empirically test whether this observation is sound, we sample some flights and see how many past predictions change upon adding a new datapoint. The longest path which changed was of length 72 with sample size $n = 8399$, with average change of 2.9 predictions in the past with standard deviation of ± 5.6.

This suggests that we can keep only the most likely paths for predictions older that roughly 100 datapoints. To accommodate this behaviour in the prediction engine, we use a fixed size queue to save dynamic tables. Once we are out of memory locations in the queue, we compute the most likely path through the array (backwards pass in Viterbi algorithm), save it and evict 10% of the oldest dynamic tables from the memory. This keeps the memory costs bounded. Eviction of a percentage of all entries in the queue avoids expensive backward pass for each timestep while still leaving the queue useful.

Predictions only using the recent history allow us to keep the memory costs bounded and also decreases the computation time per datapoint (and makes it bounded), as backwards pass through the dynamic tables in the queue takes a fixed number of steps.

5.4 Results

We test the implementation on the simulated data generated by bee flight simulator described in Section 3.1 and Section 3.2. This reveals that simulation, running on a laptop with 4 cores of 1.80GHz CPUs, takes about 0.019 seconds (standard deviation 0.0039 seconds) to process an single observation for a single bee, with memory requirements about 13 MB per bee. On the same hardware, the bee tracking software easily handles tracking of 200 simultaneously.

This test suggests that the prototype software matches the requirements (R3) and (R4), implementing precise and efficient tracking of large number

of bees with intuitive visualisation and is ready to be deployed in the real life.

6 Bayesian Cramér-Rao bounds

Following the requirement (R5), we aim to place a theoretical bound on the accuracy of predictions. To this end, we employ the Bayesian Cramér-Rao bounds from Section 2.6. Recall that the bounds are mostly straightforward to compute, except for the fact that they require an explicit form of probability density functions. As described in the Section 2.7, we can estimate it using kernel density estimation.

The theoretical bound will depend on the simplifying assumption that the bee path is following the state space model. That is, the next true bee location (at the time of the observation) depends solely on the bee location at the time of the previous observation.

Obviously, almost certainly this assumption is not valid, but we argue that it is a sensible simplifying assumption to make for our purposes. One reason for this simplification is necessity: we have no hope of analytically analysing the full beehive simulation. Furthermore, the theory on Bayesian Cramér-Rao bounds is only sufficiently developed to be tractable under this assumption. Finally, we argue that since our model makes the same assumption (Markovian assumption in the hidden Markov models), it is a fair simplification for the theoretical bounds on the performance of our model.

Furthermore, since the estimate of the probability density function depends on the sample from which it is inferred, the bound will also depend on the sample and be probabilistic in nature. We will use a shortest possible, 3 second delay between observations to provide a lower bound on all other frequencies.

To choose a suitable bandwidth matrix for kernel density estimation, we will follow the conclusions of Wand and Jones (1993) and consider only diagonal matrices. More precisely, for simplicity, we are going to use $H = diag(h_1, h_2, \ldots, h_d)$, a square matrix having value $H_{ii} = h_i$ and $H_{ij} = 0$ if $i \neq j$.

We select the best values for $h_1, \ldots h_n$ by leave-one-out cross-validation with maximum likelihood (Silverman 1986, p. 53). In each round, we take aside a single datapoint, and compute the likelihood for all the remaining datapoints. If we denote the likelihood for all datapoints $x_1, \ldots x_n$ except x_i by f_{-i}, then we seek to minimise the cross-validation objective:

$$CV(H) = \frac{1}{n} \sum_{i=1}^{n} \log f_{-i}(x_i) \tag{56}$$

This is implemented in Python statistical package `statsmodels`.

Having estimated the bandwidth H, it is not difficult to calculate the gradients using (48):

$$\nabla_{\boldsymbol{x}} \log \hat{p}(\boldsymbol{x}) = \frac{1}{np(\boldsymbol{x})} \sum_{i=1}^{n} (2\pi)^{-\frac{d}{2}} |H|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_i)^T H^{-1}(\boldsymbol{x}-\boldsymbol{x}_i)\right) H^{-1}(\boldsymbol{x}_i - \boldsymbol{x}) \tag{57}$$

Using $\log p(x,y) = \log p(y|x) - \log p(x)$, we are able to calculate all matrices $G_{11}, G_{12}, G_{21}, G_{22}$ required by (38)–(41).

Furthermore, we make use of the Monte Carlo simulation for calculation of the expectations, namely sampling m datapoints $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)$, and using

$$\mathbb{E}[h(\boldsymbol{x}, \boldsymbol{y})] \approx \frac{1}{m} \sum_{i=1}^{m} h(\boldsymbol{x}_i, \boldsymbol{y}_i) \tag{58}$$

for any function h, as required by calculation of $G_{11}, G_{12}, G_{21}, G_{22}$.

We implement the procedure in Python and run the bandwidth matrix selection and Monte Carlo simulation with the datapoints coming each 3 seconds. This leads to the root mean square error for filtering of 13.2 m with a standard deviation 1.7 m. For smoothing, the root mean square error is 11.2 m with a standard deviation of 1.5 m.

This is significantly better that the best results achieved with HMMs: 98.1 ± 15.1 m for filtering and 63.3 ± 11.9 m for smoothing (standard deviation after $\pm$ sign). Note that results for Hidden Markov models feature a rather large variance, thus the average is bound to be significantly larger that the lower bound. Also, lower bound is inexact due to various approximation detailed above. However, the results suggest that hidden Markov models do not have optimal accuracy while being acceptably close to it.

7 Conclusions

7.1 Summary

The project has achieved the desired goals outlined in the project description: we have researched and empirically compared different machine learning approaches and implemented a prototype bee tracking software, alongside providing theoretical bounds.

A biologically sound model of the bee flight has been adapted and augmented with the simulation of the ground sensors, serving as a generator for training and testing data.

We have cast the prediction problem as coregionalised regression model for Gaussian processes, predicting each component of a multi-output function as a weighted sum of latent functions. This approach gives us flexibility to easily deal with missing data while learning the correlations between radar observations and bee locations. Preemptively we have considered the optimisations and their effect on accuracy: the optimisations include tricks to avoid computationally expensive matrix recalculation and limiting the history size. For the latter, we empirically selected the best hyperparameters.

For hidden Markov models, the problem was cast as an inference of a hidden state (bee location) from observations, achieved by discretising the map to a grid. We select the hyperparameter empirically and perform domain-specific simplifications without affecting accuracy, using knowledge that the states have a natural spatial correspondence.

We have compared the performance of both models on the same data, using root mean square error as a criterion. In both filtering and smoothing, for all delays between subsequent observations, hidden Markov model outperformed the Gaussian processes, thus it was selected to be implemented in a prototype tracking software.

Further, to evaluate how our models performs compared to theoretical limit, we sought to calculate the lower bounds on the accuracy for predicting the bee location. To this end, we have calculated the Bayesian Cramér-Rao bounds for smoothing and tracking problems. This required several simplifications to our model: firstly, we assumed that the bee location evolves as a space state model, that is, current state depends only on the previous state. Also, we estimated the probability density functions by kernel density estimation techniques.

Finally, we have implemented the hidden Markov model approach in a prototype bee tracker software. Building on top of the skeleton implementation provided by Prof. Alex Rogers, we have added a prediction engine and user-friendly visualisation of predictions. The prediction engine uses compiled C++ code and multithreading to ensure high performance, while narrow interfaces via RedisDB and MongoDB keeps the code modularised and extensible. User interface displays the bee paths in an intuitive way, allowing visualisation of multiple bees simultaneously.

The software is optimised using an observation that only the recent observations affect the current prediction, ensuring the constant memory usage and time for prediction, thus allowing tracking for unlimited time. The testing indicates that software is lightweight enough to be able to handle tracking of 200 bees at the same time even on relatively low-resource machine.

The full system, including real hardware, will be demonstrated at Syngenta Jealott's Hill International Research Centre in the summer of 2017.

7.2 Reflections

The lessons learnt from this project include both practical machine learning approaches (including considerations for their deployment) and the bounds studied in the field of computational learning theory.

In addition to the theory and techniques outlined in the report, the project was also educational about the general modelling process. All modules were first built in Python and, afterwards, performance-critical parts were re-implemented in C++. While this approach proved to be largely successful, allowing rapid design and exploration of the data and models, quick prototyping diminished the reusability of the code – inflexible code for flight simulation caused a refactoring of a sizeable portion of the codebase. An advice for similar projects would be to pay attention to good OOP practices even in the modelling stage to reduce development time.

7.3 Limitations

Notably, the project has certain limitations. Firstly, as we did not have access to the real-life data, both training and testing was performed on the data generated by a beehive simulator. While the simulation is theoretically sound, it is unknown how well would the model perform in practice.

Further, the hive simulation discretises the map, thus limiting the accuracy to the size of the cell. While the root mean square error for both approaches appears to be too high for this discretisation step to make any substantial difference, it was not empirically tested.

In addition, in our sensor simulation we relied on the simple model of noise for observations' signal strength, adding white Gaussian noise. Crucially, we did not have access to the characteristics of the sensors to be used for bee tracking, thus we have provided our best pessimistic guess about the expected noise values.

All of these factors might contribute to a different behaviour of the model in the real life, increasing or decreasing accuracy. Furthermore, to make the computation feasible, we were forced to make an Markovian assumption for Bayesian Cramér-Rao bounds and also estimate the probability density functions by kernel density estimation and Monte Carlo methods. All of these simplifications contribute to inaccuracy and probabilistic nature of the lower bound.

7.4 Extensions of the project

Possible additions to the project include testing with the real life data. The prototype can be deployed in the field for prediction of real bees' movements to test the usefulness of the bee tracking software.

Further, to resolve the problem of providing a guess for the noise values and ranges of radars, we could consider comparing GPS data with the radar observations. One possibility would be to carry beacons while walking or use a drone to move the bee sensor in the field with placed ground radars while tracking the actual position of the sensor with the GPS. This would allow to infer the operational characteristics of radars and beacons.

A related extension would be to collect and use real bee flight data for both learning and testing, alleviating the concerns that results might differ in real life situations.

From the modelling perspective, different kernel functions could be tested in the Gaussian processes approach, as the choice of covariance matrix calculation may have a significant effect on accuracy.

From the theoretical standpoint, it would be useful to consider the ways to remove Markovian assumption from the Bayesian Cramér-Rao bounds computation, alleviating the concerns for inaccuracy. Further, other types of similar bounds could be used, such as Bhattacharyya, Bobrovsky-Zakai or Weiss-Weinstein lower bounds, as noted by Reece and Nicholson (2005).

References

Álvarez, Mauricio A., Lorenzo Rosasco, and Neil D. Lawrence (2012). "Kernels for Vector-Valued Functions: A Review". In: *Found. Trends Mach. Learn.* 4.3, pp. 195–266. DOI: 10.1561/2200000036. URL: http://dx.doi.org/10.1561/2200000036.

Dauwels, J. (2005). "Computing Bayesian Cramer-Rao bounds". In: *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.* Pp. 425–429. DOI: 10.1109/ISIT.2005.1523369.

Forney, G. D. (1973). "The viterbi algorithm". In: *Proceedings of the IEEE* 61.3, pp. 268–278. DOI: 10.1109/PROC.1973.9030.

Fujimoto, Richard M., George Riley, and Kalyan Perumalla (2006). *Network Simulation.* 1st ed. Vol. 0. Synthesis Lectures on Communication Networks. Morgan and Claypool Publishers.

Genton, Marc G. (2002). "Classes of Kernels for Machine Learning: A Statistics Perspective". In: *J. Mach. Learn. Res.* 2, pp. 299–312. URL: http://dl.acm.org/citation.cfm?id=944790.944815.

Ghahramani, Zoubin (2002). "Hidden Markov Models". In: River Edge, NJ, USA: World Scientific Publishing Co., Inc. Chap. An Introduction to Hidden Markov Models and Bayesian Networks, pp. 9–42. URL: http://dl.acm.org/citation.cfm?id=505741.505743.

Kawai, Reiichiro and Sergei Petrovskii (2012). "Multi-scale properties of random walk models of animal movement: lessons from statistical inference". In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 468.2141, pp. 1428–1451. DOI: 10.1098/rspa.2011.0665. eprint: http://rspa.royalsocietypublishing.org/content/468/2141/1428.full.pdf. URL: http://rspa.royalsocietypublishing.org/content/468/2141/1428.

Levin, Simon A. (1992). "The Problem of Pattern and Scale in Ecology: The Robert H. MacArthur Award Lecture". In: *Ecology* 73.6, pp. 1943–1967. DOI: 10.2307/1941447. URL: http://dx.doi.org/10.2307/1941447.

Mann, Richard, Robin Freeman, Michael Osborne, Roman Garnett, Jessica Meade, Chris Armstrong, Dora Biro, Tim Guilford, Stephen Roberts, and Paul M. Goggans (2009). "Gaussian Processes for Prediction of Homing Pigeon Flight Trajectories". In: *AIP Conference Proceedings* 1193.1, p. 360. DOI: 10.1063/1.3275635.

Mann, Richard, Robin Freeman, Michael Osborne, Roman Garnett, Chris Armstrong, Jessica Meade, Dora Biro, Tim Guilford, and Stephen Roberts (2011). "Objectively identifying landmark use and predicting flight trajectories of the homing pigeon using Gaussian processes". In: *Journal of The Royal Society Interface* 8.55, pp. 210–219. DOI: 10.1098/rsif.2010.0301.

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press, pp. 775, 515–518.

Osborne, Michael (2010). "Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature". PhD thesis. PhD thesis, University of Oxford.

Rasmussen, Carl Edward and Christopher K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Reece, S. and D. Nicholson (2005). "Tighter alternatives to the Cramer-Rao lower bound for discrete-time filtering". In: *2005 7th International Conference on Information Fusion*. Vol. 1, 6 pp.–. DOI: 10.1109/ICIF.2005.1591842.

Reynolds, Andrew M., Alan D. Smith, Randolf Menzel, Uwe Greggers, Donald R. Reynolds, and Joseph R. Riley (2007). "Displaced honey bees per-

form optimal scale-free search flights". In: *Ecology* 88.8, pp. 1955–1961. DOI: 10.1890/06-1916.1. URL: http://dx.doi.org/10.1890/06-1916.1.

Schervish, Mark J. (1997). *Theory of statistics.* Berlin; New York: Springer-Verlag Inc, p. 301.

Shih, M. C., S. Renuka, and K. Rose (2015). "2D hidden Markov model with spatially adaptive state-space for tracing many cells in image sequence". In: *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, pp. 1452–1456. DOI: 10.1109/ISBI.2015.7164150.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis.* London: Chapman & Hall.

Simonoff, Jeffrey S. (1998). *Smoothing Methods in Statistics (Springer Series in Statistics).* Springer.

Tichavsky, P., C. H. Muravchik, and A. Nehorai (1998). "Posterior Cramer-Rao bounds for discrete-time nonlinear filtering". In: *IEEE Transactions on Signal Processing* 46.5, pp. 1386–1396. DOI: 10.1109/78.668800.

Van Trees, Harry L. (1968). *Detection, Estimation, and Modulation Theory, Part I.* New York: Wiley.

Wand, M. P. and M. C. Jones (1993). "Comparison of smoothing parameterizations in bivariate kernel density estimation". In: *Journal of the American Statistical Association* 88.422, pp. 520–528. URL: http://oro.open.ac.uk/28293/.

Wikelski, Martin, Jerry Moxley, Alexander Eaton-Mordas, Margarita M. LÃşpez-Uribe, Richard Holland, David Moskowitz, David W. Roubik, and Roland Kays (2010). "Large-Range Movements of Neotropical Orchid Bees Observed via Radio Telemetry". In: *PLOS ONE* 5.5, pp. 1–6. DOI: 10.1371/journal.pone.0010738. URL: http://dx.doi.org/10.1371%2Fjournal.pone.0010738.

Williams, Ray and Paulo De Souza Junior (2016). *Swarm Sensing Modelling.* Version v2. DOI: http://doi.org/10.4225/08/57A7DE31147FA.

Wolf, Stephan, Dino P. McMahon, Ka S. Lim, Christopher D. Pull, Suzanne J. Clark, Robert J. Paxton, and Juliet L. Osborne (2014). "So Near and Yet So Far: Harmonic Radar Reveals Reduced Homing Ability of Nosema Infected Honeybees". In: *PLOS ONE* 9.8, pp. 1–15. DOI: 10.1371/journal.pone.0103989. URL: http://dx.doi.org/10.1371%2Fjournal.pone.0103989.