Master thesis of

Automatic Control and Robotics

SLAM with the Sphero robot

Xiaoxuan Hu

Supervisors:

Thomas, Federico

Porta, Josep Maria

Universitat Politècnica de Catalunya

Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

Spain

2018

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

ETSEIB

Contents

ETSEIB

List of Figures

List of Tables

Chapter 1

Introduction

The Simultaneous Localization and Mapping (SLAM) problem for mobile robots aims at building a map of an unknown environment while simultaneously determining the robot's position within this map. In the robotics community, SLAM is considered a solved problem in the most common settings [1], but an approach to the SLAM problem using minimal sensing information is still relevant both from the theoretical and practical point of view.
In this context, this M.Sc. thesis aims at studying and implementing a special type of SLAM solely based on the information provided by the on-board Inertial Measurement Unit (IMU) from a spherical mobile robot called Sphero, developed by Orbotix(see Fig. 1.1.)

Figure 1.1: Sphero robot

This IMU permits detecting impacts with the environment and performing odometry subject to significant errors. Thus, the SLAM problem will be restricted to environments defined by

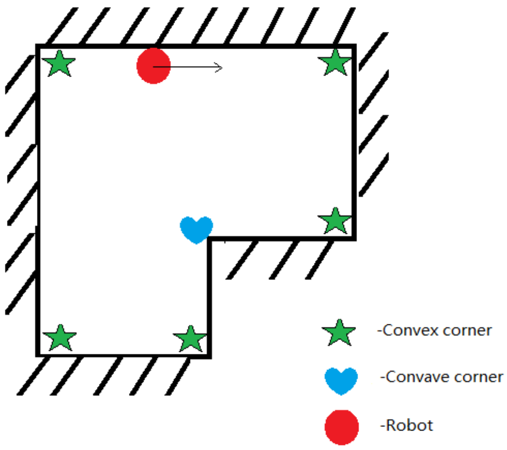

Figure 1.2: Example of the type of environments explored and mapped in this thesis.

closed regions bounded by orthogonal walls meeting at convex and concave corners (see Fig. 1.2). In this kind of environments, the robot can follow a motion in contact with the walls, while detecting corners, till it considers that it has returned to the initial point. As a result, a map of the environment is completed.

We will show how it is possible to define the location of the corners or landmarks of the rectilinear environments from changes in the robot's speed. A map representation, that consist of the location of the landmarks, their uncertainty, and their connection through straight walls, is defined. Three uncertainty models have been used and compared: a probabilistic model and two bounded-error models. Two basic operations have been implemented for the three uncertainty models: propagation and fusion.

Propagation is the basic operation that permits assigning an uncertainty to the location of a landmark, given the location and uncertainty of the previously visited landmark and the estimated error committed in the odometry. If this new landmark has already been visited, it has a previous assigned uncertainty. This uncertainty has then to be updated given the new uncertainty using a fusion operation. This concept of fusion and propagation of uncertainty applies to the three mentioned methods, they only differ in their implementation.

The probabilistic approach has dominated much of the work on low-level sensing processes handling uncertainty. Although probabilistic models which assume a uniform distribution inside a range [5] have been used, the computational tractability of low-level sensing processes, under this approach, requires the general assumption that the experimental error is simply an additive term with Gaussian distribution, and the fusion operation is essentially that of maximum likelihood estimation. Nevertheless, it is difficult to give complete error analysis because the complexity of the process of extracting low level data. Instead, when using bounded-error models, every measurement is assumed to lead to an uncertainty set in the space of parameters where the actual value is bound to be, and the fusion operation essentially reduces to find a bound for the intersection of two sets.

We will show how the two bounded-error models are of interest in those situations in which no probabilistic description of errors is available, only bound on them are known. Thus, these models are of particular interest in minimalist robotics. Actually, one important goal of this thesis has been to show how, using simple sensor information, it is possible to solve complex tasks, such as computing a map of the boundaries of an office-like environment. Thus, we have tried to answer the basic question raised by minimalist robotics: How simple can a robot be while nevertheless accomplishing interesting tasks? There are at least three important reasons to focus on minimalist robots: (1) it encourages us to find what is the least amount of information needed to solve a certain task, giving insights into the task's inherent complexity; (2) it also encourages us to use inexpensive sensors providing very limited or noisy sensing; and, finally, (3) it may allow us to manufacture inexpensive robots with low energy consumption.

This thesis is structured as follows. Chapters 2 and 3 describe the basic theory behind the mapping solutions relying, respectively, on the Kalman filter an on the set theoretic approaches. Then, Chapter 4 describes the hardware and software of the Sphero robot, including the robot's control, motion and calibration. Chapter 5 presented the implementation and the experimental results. Finally, Chapter 6 includes the conclusions and indicate possible extensions of this work.

Chapter 2

Probabilistic Approach

In this chapter, the probabilistic approach to the corner-based SLAM problem is discussed. First we introduce the notation used in this chapter, in Section 2.2 the state of art solutions are presented, along with their strong points and drawbacks. Then, Section 2.3 details the Kalman Filter solution applied in the Sphero case.

2.1 Notation

For a given time 't', the notation we are going to use is:

r_t : Pose of robot at time t

M_t : Map$\{l_1, ...l_t\}$, where l_i is the position of the $i-th$ corner.

$x_t = (r_t, M_t)$ state to estimate

u_t : Action

z_t : Observation

We use the notation $u_{1:t}$ to $z_{1:t}$ to denote the actions and the observations from time 1 to t.

2.2 Introduction to the Basic Assumptions

2.2.1 Markov assumption

Markov assumption is a concept in probability theory named after the Russian mathematician Andrey Markoff. For computational efficiency, the state transition probability assumes Markov propagation in the system state. The Markov assumption states that instead of the previous state sequence, only the previous state influence on the probability distribution of the new

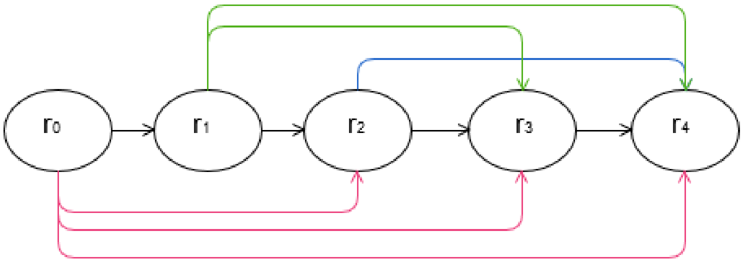

Figure 2.1: Using a general Bayesian network.

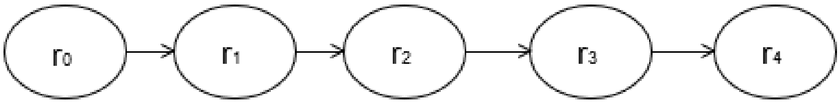

Figure 2.2: Using Bayesian network simplified by Markov assumption.

state. The expression is as follows:

$$P(r_t|r_{1:t}) = P(r_t|r_{t-1}) \tag{2.1}$$

In our project Markov assumption could simplify the probability calculation. Figure 2.1 and figure 2.2 show the sequence of Markov w2assumption. If we don't use Markov assumption, the current robot pose is based on not only the previous state but all the rest states.

2.2.2 Independent errors in actions

The errors in actions comes from the robot movement from one corner to another. It is a random value and only related to the wall length between the two corners. The action error is independent of previous robot's actions.

2.2.3 Independent errors in observations

The errors in observations comes from the detection of corners. A random observation error is obtained when the robot re-observe a corner it met before

2.2.4 Uncorrelation between actions and observation

The actions and observation is independent from each other.

2.2.5 Map is static

In the experiment, we only consider the map with static obstacles.

2.3 General Framework

The goal of this section is to estimate the probability distribution of possible maps and robot pose (i.e. positions and orientations) based on observation and control. Here the state estimation distribution in a general way is presented,

$$P(x_t|u_{1:t}, z_{1:t}, x_0) \propto p(z_t|x_t, z_{1:t-1}, u_{1:t}, x_0) \cdot p(x_t|z_{1:t-1}, u_{1:t}, x_0) \tag{2.2}$$

By applying the hypothesis in the previous section, we could rewrite the formula as below.

1. Uncorrelation between actions and observation, Markov assumption and independence between observation

$$P(x_t|u_{1:t}, z_{1:t}, x_0) \propto p(z_t|x_t) \cdot p(x_t|z_{1:t-1}, u_{1:t}, x_0) \tag{2.3}$$

2. Current state is based on integrating over all possible previous maps and states

$$P(x_t|u_{1:t}, z_{1:t}, x_0) \propto p(z_t|x_t) \cdot \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}, x_0) \cdot p(x_{t-1}|z_{1:t-1}, u_{1:t}) d_{x_{t-1}} \tag{2.4}$$

3. The map is static

$$P(x_t|u_{1:t}, z_{1:t}, x_0) \propto p(z_t|x_t) \cdot \int p(r_t|x_{t-1}, z_{1:t-1}, u_{1:t}, x_0) \cdot p(r_{t-1}|z_{1:t-1}, u_{1:t}) d_{x_{t-1}} \tag{2.5}$$

4. Conditional probability and Markov assumption

$$P(x_t|u_{1:t}, z_{1:t}, x_0) \propto p(z_t|x_t) \cdot \int p(r_t|r_{t-1}, u_t) \cdot p(r_{t-1}|z_{1:t-1}, u_{1:t}) d_{r_{t-1}} \tag{2.6}$$

In this equation we can see that, the current state distribution $P(x_t|u_{1:t}, z_{1:t}, x_0)$ depends on the observation model, the motion model and all the previous states estimation. The integral in equation 2.6 corresponds to a propagation operation and product with the observation model is a fusion operation. This recursive estimation can be implemented in several ways, detailed next.

2.4 State of Art

In the different SLAM approach, we represent the uncertainty and implement the propagation and fusion operation in different ways.

2.4.1 Kalman filter

The Kalman filter method is based on Gaussian a representation of the uncertainly. It is widely used especially in case where the system has random perturbations or there exist white noise in the source of measurement. In linear systems, it produces the best estimation given the system model and the measurement. But in the Kalman filter theory, it is assumed that the measured and estimated noise are all white noise and conforms to a Gaussian distribution. Such assumption, constrain the usage of the method.

2.4.2 Information filter

From the analytical viewpoint, the information filter is similar to the Kalman filter, the only difference is that instead of estimating the covariance, the information filter use the information matrix, which is the inverse of the covariance. The advantage of information filter is that the information matrix is almost sparse and thus the computational complexing can be reduced by using sparse approximation of the information matrix.

2.4.3 Pose SLAM

In the pose SLAM, the robot trajectory is the only parameter estimated, the landmarks are only used as a tool to generate the relative constraints among the robot poses. In addition, observations appear in the form of relative motion measurements from robot pose. The mapping approach presented in this thesis can be seen as a pose SLAM solution since the features in the map are actually previous poses of the robot.

2.4.4 Particle filter

In the particle filter method, the estimated robot pose is represented using particles. For instance, in [7], a Rao-Blackwellized particle filter is used. In this study, we could see there exists some problem in the particle filter. The computational efficiency is relatively low because the number of the particles grows exponentially in high-dimensional spaces with the dimension of the state space and thus it is hard to obtain accurate solution.

2.5 Particularize the derivation for Kalman case

Although the Kalman filter approach has weakness, we decide to use it due to its simplicity and because our model are linear. When we particularize the general framework to the Kalman filter case, one extra assumption is needed. It is necessary to assume the probability distributions for all states are Gaussians. Which could be presented as,

$$P(x_t|u_{1:t}, z_{1:t}, x_0) \sim N(\mu_x, \Sigma_x), \tag{2.7}$$

where N is a normal distribution, μ_x is the mean (robot pose) and Σ_x is the covariance, represents the noise corresponding to that position.

2.5.1 Motion model

The robot motion model could be described as,

$$P(r_t|r_{t-1}, u_t, x_0) = f_r(r_{t-1}, u_t) + w_t, \tag{2.8}$$

where f is a generic time-update movement function, w_t is the motion noise that $w_t \sim N(0, Q_t)$ Since f only affects r in x, for convenience, we can also write the formula as,

$$P(x_t|x_{t-1}, u_t, x_0) = f_x(x_{t-1}, u_t) + w_t, \tag{2.9}$$

when the state changes, the prediction step is the motion model updates the current Gaussian [11]. As,

$$\begin{cases} \hat{\mu}_t = f_x(\mu_{t-1}, u_t) \\ \hat{\Sigma}_t = \nabla f_x \cdot \Sigma_{t-1} \cdot \nabla f_x^T + Q_t \end{cases} \tag{2.10}$$

Where $\nabla f = \frac{\partial f_x}{\partial r} = (\frac{\partial f_x}{\partial r}, \frac{\partial f_x}{\partial m}) = (\frac{\partial f_r}{\partial r}, 0)$.

2.5.2 Observation model

There are two possibilities when robot observing one landmark. One possibility is that the robot re-observes a known landmark. The loop should be closed at this time. The robot position, map and covariance should be updated according to this observation. Another possibility is that the robot observes a new landmark. Then the new landmark should be added in the states. In this section, the two possible cases are presented.

The robot re-observes a known landmark

The observation model could be written as,

$$z_t = h(x_t) + v_t, \tag{2.11}$$

where h is a generic time-update observation function, v_t is the noise in measurement, $v_t = N(0, R_t)$.
When re-observing a known landmark, the states will be re-updated taking into account the closure loop state (current robot pose). The updates of the mean and covariance at time t could be described as,

$$\begin{aligned} \mu_t &= \hat{\mu}_t + W_t \cdot (z_t - h(x_t)), \\ \Sigma_t &= \hat{\Sigma}_t - W_t \cdot S_t \cdot W_t^T, \end{aligned} \tag{2.12}$$

where W_t is the Kalman gain, $W_t = \hat{\Sigma}_t \cdot \nabla S_t^{-1}$ and $z_t - h(x_t)$ is the error in observation. $S_t = \nabla h \cdot \hat{\Sigma}_t \cdot \nabla h^T + R_t$, and $\nabla h = \partial h / \partial x$.

The robot observes a new landmark

When the robot finds a landmark which not yet been mapped, the landmark initialization is needed. Here the inverse observation model could be formed as,

$$l = g(x_t, z_t). \tag{2.13}$$

Where l is the position of landmark, x_t is the state(only r_t), z_t is the observation.
The formula is update,

$$\begin{aligned} \mu_x &= (\hat{\mu}_x, l) \\ \Sigma_x &= \begin{bmatrix} \hat{\Sigma}_t & \Sigma_{xe}^T, \\ \Sigma_{xe} & \Sigma_e, \end{bmatrix} \end{aligned} \tag{2.14}$$

where $\Sigma_{xe} = \nabla g_x \Sigma_x$, $\Sigma_e = \nabla g_x \Sigma_x \nabla_{gx}^T + \nabla g_z R_t \nabla_{gz}^T$, $\nabla g_x = \frac{\partial g}{x_t} = (\frac{\partial g}{\partial r_t}, \frac{\partial g}{\partial M_t}) = (\frac{\partial g}{\partial r_t}, 0)$, $\nabla g_z = \frac{\partial g}{\partial z_t}$

2.5.3 Data association

We re-observe a landmark l_i for a given norm$\| \cdot \|$ and threshold s if

$$\|g(x_t, z_t) - l_i\| < s. \tag{2.15}$$

In the Kalman filter approach the used distance is Mahalanobis distance:

$$(\mu_i - g(x_t, z_t))^T \cdot [\Sigma_i + \Sigma_e]^{-1} \cdot (\mu_i - g(x_t, z_t)) < s, \tag{2.16}$$

where $\mu_i$is mean position of l_i, $g(x_t, z_t)$ is the observation model, $\Sigma_e = \nabla g_x \Gamma_x \nabla_{gx}^T + \nabla g_z R_t \nabla_{gz}^T$ and Σ_i is the block diagonal taken from Σ_x. In experiment, we tune the threshold s to use Mahalanobis distance to identify if the robot re-observes a landmark. If the Mahalanobis distance is bigger than a threshold, we consider it is a new landmark and we do the landmark initialization. On the contrary, we consider that the robot re-observes a known landmark, it closes the loop, and corrects the map.

2.6 Application to the Sphero robot

In this section, we particularize the Kalman filter approach for Sphero robot. The filter will be used to estimate the pose of the robot and the map will be used to improve the noisy estimations produced by the dead-reckoning algorithm. The correction is applied each time the robot reaches a corner already in the map.

In the corner-based SLAM problem, the robot Sphero slides along the wall (with robot orientation θ) in the given rectilinear environment until it meets the corner, then rotates itself according to the corner type. In figure 2.3 we show an simple environment with robot orientation angle. The red ball is the robot, the yellow squares are the corners(here we only show convex corners), the X and Y are the axes in the world frame, θ is the robot orientation (the angle between the robot heading and the X axes in the world frame). First of all, we introduce the notation in this section:

1. $r_t = (x_t, y_t)$ Pose of robot at time t
2. $l_i = (x_i, y_i)$ Pose of Landmark i (corner i)
3. $M_t = (l_1 \cdots l_t)$ Landmarks(Map), corners seen so far
4. $r_{t+1} = f(r_t, u_t)$ Motion model

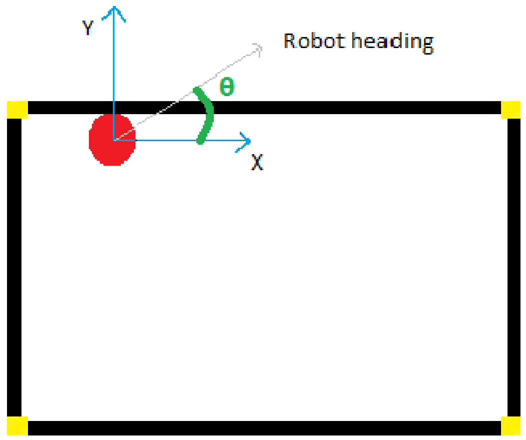

Figure 2.3: Slide mode and simple rectilinear environment.

2.6.1 Map Description and initialization

In this situation, the map estimation could be described as $M = \{\mu, \Sigma\}$ Where μ is a $2 \times t$ matrix which contains all the pose of landmarks and $\sum$ is a $2t \times 2t$ matrix representing the robot's and landmark's noise and the cross-covariances between them.

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_t \end{bmatrix} \tag{2.17}$$

$$\Sigma = \begin{bmatrix} \Sigma_{L_1L_1} & \cdots & \Sigma_{L_1L_t} \\ \vdots & \ddots & \vdots \\ \Sigma_{L_tL_1} & \cdots & \Sigma_{L_tL_t} \end{bmatrix} \tag{2.18}$$

Where $\mu_i = (x_i, y_i)$ is the position of the i_{th} corner. The matrix Σ is symmetric, $\Sigma^T = \Sigma$.

2.6.2 Robot motion model

The on-board accelerometer sensor continuously tracks the acceleration of the Sphero robot in both X and Y direction. The coordinates of the robot in both X and Y axes from the initial value $[0, 0]$, are available through API from on-board odometer computed by dead-reckoning method. From one corner to another, the robot starts with the initial velocity equals to zero. So, the increment of the coordinates is

$$\Delta x = \sum_{i}^{n} a_x \cdot \delta t_i + \frac{1}{2} \cdot a_x \cdot cos(\theta) \cdot \delta t_i^2, \tag{2.19}$$

$$\Delta y = \sum_{i}^{n} a_x \cdot \delta t_i + \frac{1}{2} \cdot a_y \cdot sin(\theta) \cdot \delta t_i^2, \tag{2.20}$$

where Δx and Δy are the coordinates increment from one corner to another in X and Y axes, δt is the sample time, a_x and a_y are the mean acceleration during the sample time δt.
From here the movement of the robot can be simply described as a unicycle. We could use two specific model to analyze the system. One is a speed based model, and another is a distance based model. Both model are based on the discrete time events and they are adequate to describe the motion and deal the un-modeled dynamics and uncertainties. Speed model use speed and heading (angle) as input, output is the calculated Map and trajectory. Distance based model use the odometry from Sphero robot, the input for our model is the incremental part of odometry, in Sphero case an assumption is displacement of the odometry, and output is still the Map and trajectory. We know that when we model the system, we cannot get a deterministic model. We need to take into account the uncertainty in the system. There is a subtle connection between model complexity and system uncertainty. If we use a more complete model (more complex), there will be less underlying phenomena we didn't consider, the system will behave more similar to the real situation. Otherwise, we will have more uncertainty of the system. But from another point of view, the more complex model require a higher computational complexity. When it comes to the probabilistic robot model, the most important thing is to get an accurate model of the uncertainties both in the robot movement and observation while minimize the computational complexity.

Speed based model

As we could get the velocity from on-board sensor, one possible way is to the velocity and calculated robot pose (position and orientation) as the state according to the sample time, from time to time, we renew the robot pose using the controls from the robot based frame by applying several sequences of rotation and translation approaches. In the speed based model, we assume the motion data is from the speed and angle data given to the Sphero robot. The velocity (in centimeter) and the heading (θ in degree) are the inputs for Sphero robot when we control (see more in Section 4.2). This one requires a more accurate sensor and shorter sample time.

Distance based model

Because the Sphero robot moves from corner to corner, the output of this module for sphero robot is a vector $(\Delta x, \Delta y)$ with the estimated motion distance between the two corners. Since the walls are aligned with X or Y, we know that the actual movement is either $(\Delta x, 0)$ or $(0, \Delta y)$, so the state simply is the position of the robot. $S = (x, y)$, where x and y are the coordination of the robot position. Here our model is simpler and we have more toleration to the poor quality sensor. In addition, get distance data directly from IMU board

We compared the two different models, the speed based model is more reasonable to describe the reality, but considering the sensor we have is only the accelerometer, the distance based model is less computational complex and easier to achieve. In the practical point of view, the accuracy of the motion model is not very relevant, the distance based model is sufficient to accurately describe the dynamic of the motion. So here we choose the distance based model to the later stage. A more complex and accurate model may be considered as the future work.

2.6.3 Prediction

Once we hit a corner or detect there is an ending wall, we update the state at that point. The prediction only affects part of the state and covariance, we update the robot pose at time $t+1$ as,

$$r_{t+1} = r_t + u_t \tag{2.21}$$

Because the distance based model is considered, the u_t could be represented as,

$$u_t = \begin{cases} (\delta x, 0) & x \quad direction \\ (0, \delta y) & y \quad direction \end{cases} \tag{2.22}$$

Where the displacement Δx and Δy are actual movement described in section 2.6.2.

2.6.4 Observation

When the current robot position is associated with one landmark through data association, the observation is the current robot position could be described as the function,

$$z_t = h(x_t) = r_t, \tag{2.23}$$

where z_t is the observation, x_t is the current state $x_t = (r_t, M_t)$ and h is the observation function which is linear. This equation could be simplified from equation 2.12 since $\nabla h = \frac{\partial h}{x_t} = (\frac{\partial h}{\partial r_t}, \frac{\partial h}{\partial M_t}) = (I_2, 0)$. I_2 is a identity matrix of size 2. Then by applying the Kalman filter approach, the current robot position, landmarks position and covariance are all updated by the correction.

Because the current robot position is always the same as a corner position, so if the robot reaches a new corner it didn't meet before, it will be added to the map through landmark initialization. This procedure uses the inverse observation model,

$$l = g(x_t, z_t) = r_t, \tag{2.24}$$

where l is a new landmark, g is the inverse observation model, z_t is the observation. This equation is simplified from equation 2.14 since $\nabla g_x = \frac{\partial g}{x_t} = (\frac{\partial g}{\partial r_t}, \frac{\partial g}{\partial M_t}) = (I_2, 0)$, $\nabla g_z = 0$. At the time robot meets a new corner, both the landmarks and covariance increase its size to $n+1$ add the new observation of the new landmark at current robot pose, $l = r_t$.

Chapter 3

Set Theoretic Approach

Normally we always model the uncertainty as Gaussian white noise. But in real world, we know there exists a lot of uncertainty which is non-Gaussian, non-white noise and also systematic errors. In the set theoretic approach, we explain the uncertainty as a bounded region(i.e., an ellipsoid, a bounding box, ect.) [9]. Those bounding region could easily explain the uncertainty of the estimate position in the SLAM problem.

There are already several applications based on set theoretic approach related to SLAM problem. One is Cuik-SLAM [8], it based on an interval-based kinematic method to formalize a SLAM problem, the constrains even nonlinearities could be modeled effectively because of the structure imposed by the motion and sensing capabilities of the robot. In [3], a set-valued methods to solve the localization and mapping problem is introduced. Also, Jaulin [6] proposes a interval analysis way of set membership method as another SLAM solution in set theoretic approach. They convert the SLAM problem to a constraint satisfaction problem, using propagation method and test the approach in an underwater robot.

Here we proposed another set theoretic approach to solve the SLAM problem. By using different methods to model the uncertainty, we use propagation to model the robot movement and and fusion to achieve data association and loop closure state update

In Section 3.1 we give the basic definitions related to this chapter. In Section 3.2 some state of art are discussed. Then, in Section 3.3 we apply the set theoretic approach to the case of SLAM with the Sphero.

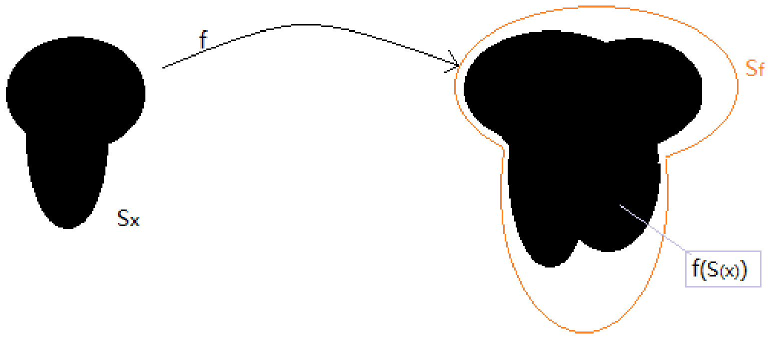

Figure 3.1: Propagation operation

3.1 Basic Definitions

3.1.1 Uncertainty representation

In set theoretic approach, uncertainty is described as a bounded region. The region could be represented as a set $p(x) = \frac{1}{|S|}$. Where $|S|$ is the volume of set S, p is the probability, x is a parameter, in our case is the robot position. The probability of all x in the set S equals 1, i.e $\int_s p(x) \cdot dx = 1$. We could assume the true position of robot will be anywhere in this region with the same possibility.

3.1.2 Propagation operation

The propagation is the estimation of the effect of a function on variables. The propagation of uncertainty for a given function f and set S, if $x \in S_x$ then $f(x) \in S_f$, where S_f is the propagation result, an approximation of the actual set $f(x)$.
The figure 3.1 shows the propagation operation. $f(x)$ is the actual set of propagation and S_f is the approximation of that set. Different methods may lead to different accuracies in the approximation of a given function.

3.1.3 Fusion operation

Fusion operation estimates the common area of two different set. We assume S_x, S_y are the two sets, S_{xy} is the fusion result of those two different sets, we could $S_x \cap S_y \subset S_{xy}$. Which could

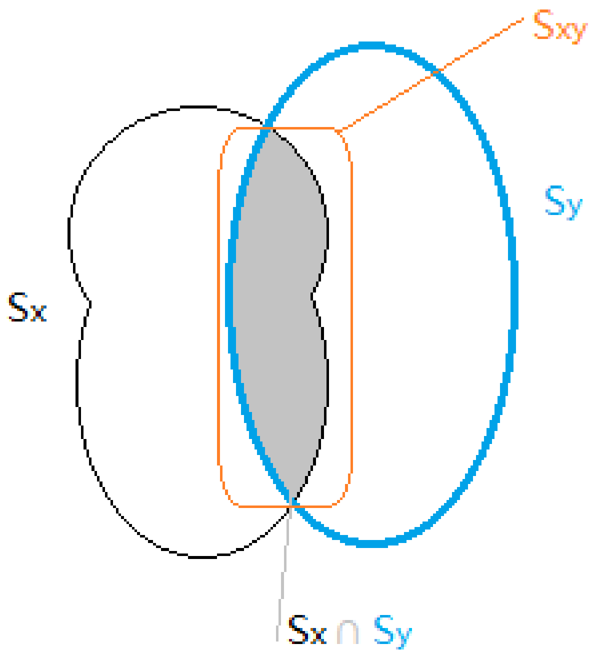

Figure 3.2: Fusion operation

be described by figure figure 3.2. Different methods in fusion operation will provided different accuracies on the approximation.

3.2 Set theoretic approaches

We present two set theoretic approaches in this section, one is the ellipsoid approach and another is the bounding box approach. Here we propose two different way to represent that range. One is ellipsoid, another is bounding box.
For such approaches, the set representation and the volume of each set in 2-D space is present. Then, the particular propagation and fusion operators are detailed.

3.2.1 Ellipsoidal approach

The 2-D Ellipsoid could be defined by a vector and a matrix,

$$x \sim \xi_0(x_0, E_0) \tag{3.1}$$

Where x_0 is a vector 2×1, $x \in \Re^2$ and E_0 is a semi-definite matrix 2×2. The points inside this region S_x are

$$S_x = \{x|(x - x_0)^T \cdot E_0 \cdot (x - x_0) \leq 1\}. \tag{3.2}$$

The volume of the ellipsoid is,

$$V = \frac{v_n}{\sqrt{|E|}}. \tag{3.3}$$

Where v_n is the volume of unit circle in $\Re^2$. In some sense the E here is the inverse of the covariance matrix Σ in Kalman filter. In probabilistic approaches, Σ^{-1} is devoted by Λ, the information matrix, which is equivalent to E here.

Minkowski sum and difference of ellipsoids

Minkowski sum and difference of ellipsoids operations are introduced here since they are later used in the fusion operation. It assumed that we know two ellipsoid x and y where $x \in \xi(x_0, E_1)$, $y \in \xi(y_0, E_2)$, the Minkowski sum of those two could be described as $z = x \oplus y$, could be formed into,

$$(I_2, I_2)v - z = 0 \tag{3.4}$$

where v is associated with the uncertainty region, $v = (x, y) = \begin{pmatrix} x \\ y \end{pmatrix}$. Since E_1 and E_2 are nonsingular matrices, $rank(E_1) = rank(y) = 2$, we could easily know that $rank(v) = rank(x) + rank(y) = 4$.
By applying the ellipsoid fusion formula [10], we could get,

$$\begin{aligned} &v \in \xi_v(v_0, E_0) \qquad v_0 = (x_0, y_0) \\ &E_0 = \begin{pmatrix} E_1/2 & 0 \\ 0 & E_2/2 \end{pmatrix} \end{aligned} \tag{3.5}$$

Minkowski difference of ellipsoids is similar to the Minkowski sum, if we want to get the difference between $x \in \xi_x(x_0, E_1)$ and $y \in \xi_y(y_0, E_2)$, we could get it from Minkowski sum of $x \in \xi_x(x_0, E_1)$ and $y \in \xi_y(y_0, E_2)$. Will be,

$$\begin{aligned} &v \in \xi_v(E_0, v_0) \\ &v_0 = (x_0, -y_0) \\ &E_0 = \begin{pmatrix} E_1/2 & 0 \\ 0 & E_2/2 \end{pmatrix} \end{aligned} \tag{3.6}$$

Propagation of ellipsoids

The new estimate pose of robot is the movement from the previous pose through the motion, the robot pose in time t could be expressed as r_t ,

$$\hat{r}_{t+1} = r_t + w_t, \tag{3.7}$$

where $r_t \sim \xi(r_t, E_{r_t})$, $w_t \sim \xi(0, E_{w_t})$, Here in our case the motion is linear, $\hat{r}_{t+1} = f(r_t, E_t)$. For ellipsoidal approach, the propagation is based on the Minkowski sum/difference, here first we define the linear function is $Ax + Cy + d = 0$. $y \sim \xi_y(y_0, F)$. In our case $y = r_t \sim \xi_{t+1}(\hat{r}_{t+1}, F)$ In our case, d is 0, C is $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and A is $\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$, which means,

$$\hat{r}_{t+1} = r_t + w_t \Leftrightarrow (I_2, I_2) + \begin{bmatrix} r_t \\ w_t \end{bmatrix} \tag{3.8}$$

By applying the Minkowski sum, we could easily get $\hat{r}_{t+1}$ would be expressed as,

$$\hat{r}_{t+1} \sim \hat{\xi}_{r_{t+1}}([r_t, E_t], \begin{bmatrix} E_{r_t}/2 & 0 \\ 0 & E_{w_t}/2 \end{bmatrix}) \tag{3.9}$$

Because $F = C^T G C$, where $G = (A^{\#})^T \cdot E_t \cdot A^{\#} - (A^{\#})^T \cdot E_t \cdot N_A \cdot (N_A^T \cdot E_t \cdot N_A)^T \cdot N_A^T \cdot E_t \cdot A^{\#}$, $C = -I$. and $A^{\#} = A^T(AA^T)^{-1}$. We simplify the F formula we could get,

$$F = G = 1/8(E_{r_t} + E_{m_t}) - 1/8(-E_{r_t} + E_{m_t})(E_{r_t} + E_{m_t})^{-1}(-E_{r_t} + E_{m_t}) \tag{3.10}$$

Then we implement the new robot pose. We will get,

$$\begin{aligned} &\hat{r}_{t+1} = r_t + m_t \\ &F = G = 1/8(E_{r_t} + E_{m_t}) - 1/8(-E_{r_t} + E_{m_t})(E_{r_t} + E_{m_t})^{-1}(-E_{r_t} + E_{m_t}) \end{aligned} \tag{3.11}$$

Fusion of ellipsoids

The intersection of two ellipsoids determines common range of two ellipsoids. The fusion is to find a minimum volume ellipsoid to describe that common range. For example $x \in \xi_x(x_0, E_1)$

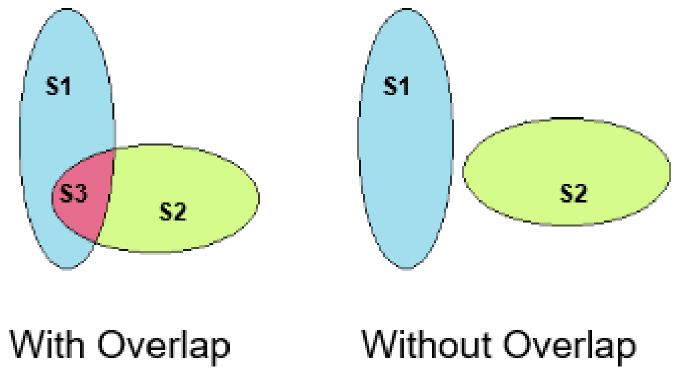

Figure 3.3: Ellipsoid with overlap and without overlap

and $y \in \xi_y(y_0, E_2)$ are two ellipsoid, we explain get the fusion $f \in \xi_f(x, E)$in the following way,

$$k(|X|)tr([X](E_1-E_2))-n(|X|)^2 \times (2x^T E_1 x_0 - 2x^T E_2 y_0 + x^T (E_2 - E_1)x - x_0^T E_1 x_0 + y_0^T E_2 y_0) = 0 \tag{3.12}$$

Because in our case, $rank(E_1)$ and $rank(E_2)$ are full rank, $rank(E_1)+rank(E_2) = rank(E_1, E_2) = 2$, so $f \in \xi_f(x, E)$ could be written as,

$$\begin{aligned} &x = (E_1^2 + E_2^2)^{-1}(E_1^2 x_0 + E_2^2 y_0) \\ &E = \tfrac{1}{2}E_1 + \tfrac{1}{2}E_2 \end{aligned} \tag{3.13}$$

The fusion operation in ellipsoid case is illustrated in figure 3.3. The S_1, S_2 are the two ellipsoids and S_3 is the intersection between the two area S. If $S_3 = \varnothing$, which means there is no overlap of S_1 and S_2, the fusion is $\varnothing$.

Data association for ellipsoidal approach

Data association is to determine the best match. In set theoretic approach it is based on uncertainty overlap. In our situation, W will be a score giving the similarity: the higher W, the higher the similarity of the landmarks.

Because it is hard to calculate the area of S_3 directly, we use fusion to get one ellipsoid S_4 with minimum volume (V_4) which could tightly bounds the intersection part (S_3) of the two given ellipsoids (S_1 and S_2, which volumes are V_1 and V_2).

Here we could define the score W as,

$$W = max(\frac{V_4}{V_1}, \frac{V_4}{V_2}) = \frac{V_4}{min(V_1, V_2)}. \tag{3.14}$$

Here we use ratio of intersection to the ellipsoid / bounding box instead of using the area directly because the uncertainty range of initial position (first landmark) is very small. If the robot meets the first landmark, the intersection between the current position and first landmark will be very small too.

The score W is between $[0-1]$, $W = 0$ means there is no overlap, $W = 1$ means the overlap is 100%, one set is included in another. If W is above a certain value, we consider the two landmarks are the same.

3.2.2 Bounding box approach

In this approach, we describe the error sets S_x as a box. The set will always be constrained in a lower and upper bound,

$$S_x = \{x | \forall_{i=1...n} \quad l_i \leq x_i \leq u_i\}, \tag{3.15}$$

x_i is the i_{th} component of x, l_i is the lower bound, u_i is the upper bound. S_x can also be represented with (l, u), where $l = (l_1, ..., l_n)$ and $u = (u_1, ..., u_n)$. The volume of a set is,

$$V = \prod_{i=1}^{n} (u_i - l_i). \tag{3.16}$$

Propagation and fusion operation

For the bounding box approach, the propagation and fusion are simple. We only need to adapt the length and width by sum or subtract from the previous bound to get the new one. The propagation could be described by,

$$\begin{aligned} S_f &= (l_i^f, u_i^f) \\ l_i^f &= min(f_i(x)) \\ u_i^f &= max(f_i(x)). \end{aligned} \tag{3.17}$$

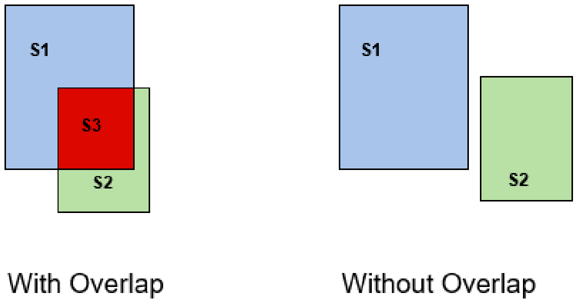

Figure 3.4: Bounding box with overlap and without overlap

Fusion operation could be formed as,

$$\begin{aligned} &S_{xy} = (l^{xy}, u^{xy}) \\ &l^{xy} = max(l^x, l^y) \\ &u^{xy} = min(u^x, u^y). \end{aligned} \tag{3.18}$$

where min/max are applied element wise.
Fusion operation for bounding box is illustrated in figure 3.4 The S_1, S_2 are the two bounding boxes and S_3 is the intersection between the two area S. The fusion of two bounding box could be $\varnothing$ as well.

For data association in the bounding box approach, the equation is the same as Eq. 3.14. The only difference is that the volume of the set is computed using Eq. 3.16 instead than Eq. 3.3.

3.3 Set Theoretic Applied SLAM with Sphero

In Sphero case, we use a graph with nodes to represent the map, where at each node is a set (ellipsoid/box) representing the corner coordinates. Each edge in the graph has the information about the displacement (the parameters of the function f in propagate step from one corner to the next). The 'back-propagation' process as a backward exploration of the graph until all nodes are visited propagating using f^{-1} (i.e. the inverse of the function f stored in the edges). We could use flowchart to display the genera idea of the strategy applied in Sphero case as figure 3.5,

In the Sphero SLAM case, from the initial point, the robot position and error are propagated through the motion model to generate a new robot pose with the ellipsoid uncertainty

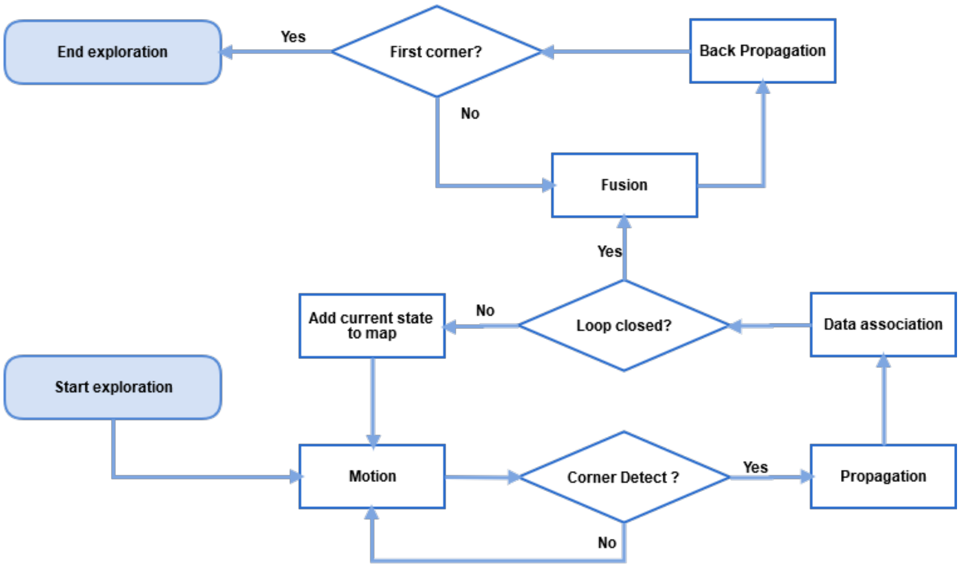

Figure 3.5: the genera idea of Set Theoretic Approach in Sphero SLAM problem

region, which is similar to the 'prediction' in probabilistic approach. Then, every time when the robot arrive at a new state, by applying the data association, we check if loop is closed. If not, we continue the movement and the propagation. If yes, we stop the robot and do back propagation, then apply fusion to each state in order to reduce the error.

To be more specific, the Sphero robot will always move along a wall. When it detect there is a corner, it applies the propagation step. The current estimation of the robot pose (the set) changes according to function f representing the displacement. The following step is data association. The current robot position will be saved and checked if it is a new corner (landmark) or it is one that already in the map. If the corner is a new one, that corner will be added in the map, the robot will be rotated to continue following another wall. If the corner is already in the map, the back-propagation will be performed with the fusion operation.

Data association in this case is the one explained in Section 3.2.1. The similarity of two sets could be calculate through formula 3.14, where the volume is computed using Eq. 3.3 and Eq. 3.16 respectively. Then, data association could be easily identified by a threshold.

Chapter 4

Sphero

4.1 Hardware of Sphero

Sphero is a ball-shaped robot of 74mm of diameter designed by Ian Bernstein and produced by the company Orbotix. This company already made two version of the Sphero, the Sphero 1.0 and Sphero 2.0. Both versions could be controlled via bluetooth by a smartphone or a tablet running Android, IOS or a Windows phone. The users can program the robot through an app named Macrolab, using a C-based language macros or orbBasic. For this reason, this robot could be easily used by children to learn coding or play games. Also it safe for children because it has no sockets either small parts on the shell, the battery are inside the ball and use wireless charge through a charging base, shown in figure 4.1. The outlook of the two version of Sphero

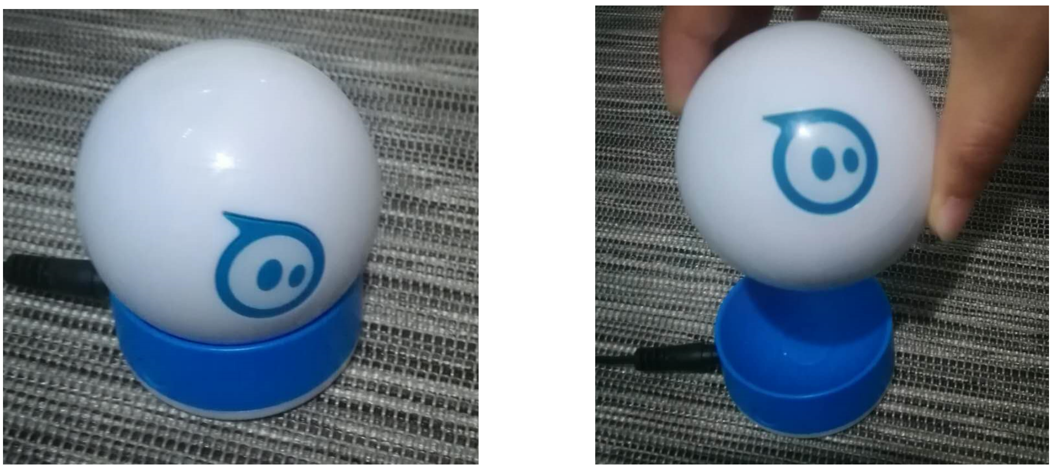

Figure 4.1: Wireless charging of Sphero robot

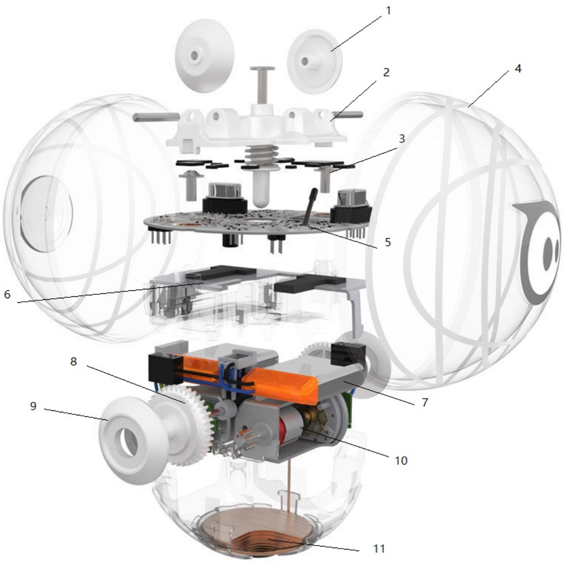

Figure 4.2: Inside of Sphero robot. See Table 4.1 for a description of the components

are the same. The robots are covered by a 3-D printed shell which protect the rolling device and the sensors inside. The robot itself is waterproof, which helps the robot easily to move on a variety of complex terrains. ,

The robot has a very compact internal composition. The internal mobile robot touches the shell with four rubber wheels. Two motors drive the motion of the bottom two wheels respectively, controlled by a 75 MHz ARM Cortex processor on board.The inside component are listed in Table 4.1.

The appearance of the Sphero 2.0 looks like the previous generation of products, but in fact, the internal components have been rearranged: the motion transfer system has been enhanced to increase the efficiency. A new user interface is provided in Sphero 2.0, the user could use it to move the ball with different speeds or change the LED light color.

The Sphero 2.0 is the enhanced version of the Sphero 1.0, the led light inside is brighter, it could roll two times faster (could reach about $2m/s$), and the bluetooth range is about $15m$. In addition, compared to Sphero 1.0, Orbotix also highly optimized the sensor, in order to achieve more accurate control operation.

The inside sensor are only two: a accelerometer and a gyroscope. The gyroscope is used to self-balance inside robot (using a PID controller). The accelerometer is used to measure the acceleration, calculate the velocity, and to estimate the distance the ball travels. In the API, we could only access the accelerometer to have the data with the sample frequency $2Hz$. It

Table 4.1: Inner parts list of Sphero robot

No.	Name	Quantity
1	Passive wheel	2
2	Strut	1
3	Plastic gaskets and screws	-
4	3-D printed Polycarbonate shell	1
5	ST STM32 F3 microcontroller	1
6	Support frame	1
7	HUATAI HT6292 Battery	2
8	Gear	2
9	Active Wheel	2
10	Motor	2
11	Wireless charging receiver	1

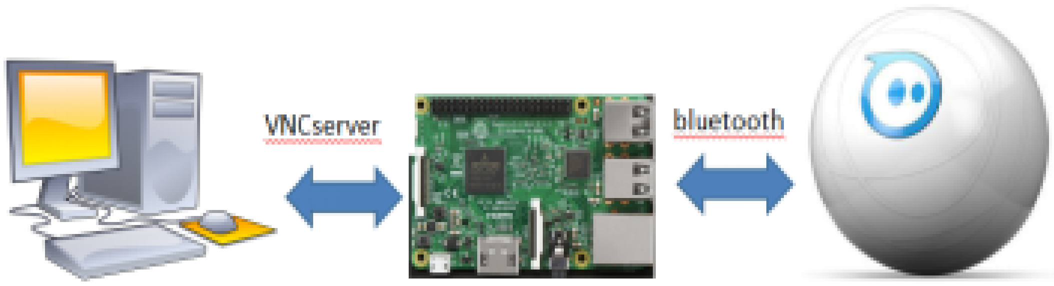

Figure 4.3: Sphero connection through VNC server

is a great challenge to use only this robot to study the SLAM problem without exteroceptive sensor, it is different from the common SLAM settings.

4.2 Sphero Control

4.2.1 Connection to Computer

Connect through a Raspberry Pi 3

As a first step, we used a Raspberry Pi 3 to connect the Sphero robot to a computer. Raspberry is used as intermediary for the purpose of communication. Since they already have the apps developed for Sphero, we tried to use it to get access to Sphero via Bluetooth in order to send signal for control, the basic idea is as figure 4.3. Here we followed the process in [4] We first installed the Node.js on Pi3 and Sphero SDK in the computer. Then we connected raspberry Pi and Sphero by the address in VNC-server. With this setting, we could connect to Sphero

with javascript and code to control the movement of Sphero.
However this approach caused several problems. The JavaScript is client a side language which executes in a browser. In order to manage it we used a Node.js, which is a JavaScript runtime built on Chrome's V8 JavaScript engine. It is a more suitable language for developing apps for the Android devices, but not for our project. In addition, there are only few pre-defined scripts where available. It would be a complex programming if we want to do experiment through this connection, that complex programming work is out of the scope of this project.
So, despite devoting a lot of effort and time to this approach, we had finally to abandon it.

Connect through Matlab Interface

An alternative way to control Sphero is using MATLAB. We found three recent works that focus on the interface to connect sphero using MATLAB:

1. Sphero MATLAB Interface

2. Sphero API MATLAB SDK

3. Sphero Connectivity Package

We tried all of them. The Sphero connectivity package is the more powerful tool. It is the newest version of sphero connection package to Matlab and it based on the instrument control box.
The package is mainly based upon a "sphero" class, which in turn relies on the MATLAB Bluetooth class. The class methods and properties allow us to perform (within MATLAB) many operations available with the underlying Sphero API, such as connecting, disconnecting, sleeping, changing LED colors, reading (and/or streaming back) the Sphero's position and velocity, and commanding each of the 2 motors independently. An higher-level roll command can also be used to move the Sphero with a certain speed and direction. The Simulink library contained in the package also features Simulink blocks for setup, timing, and basic sensing and actuation. With this package, we could fully access to the sensor data and control robot.

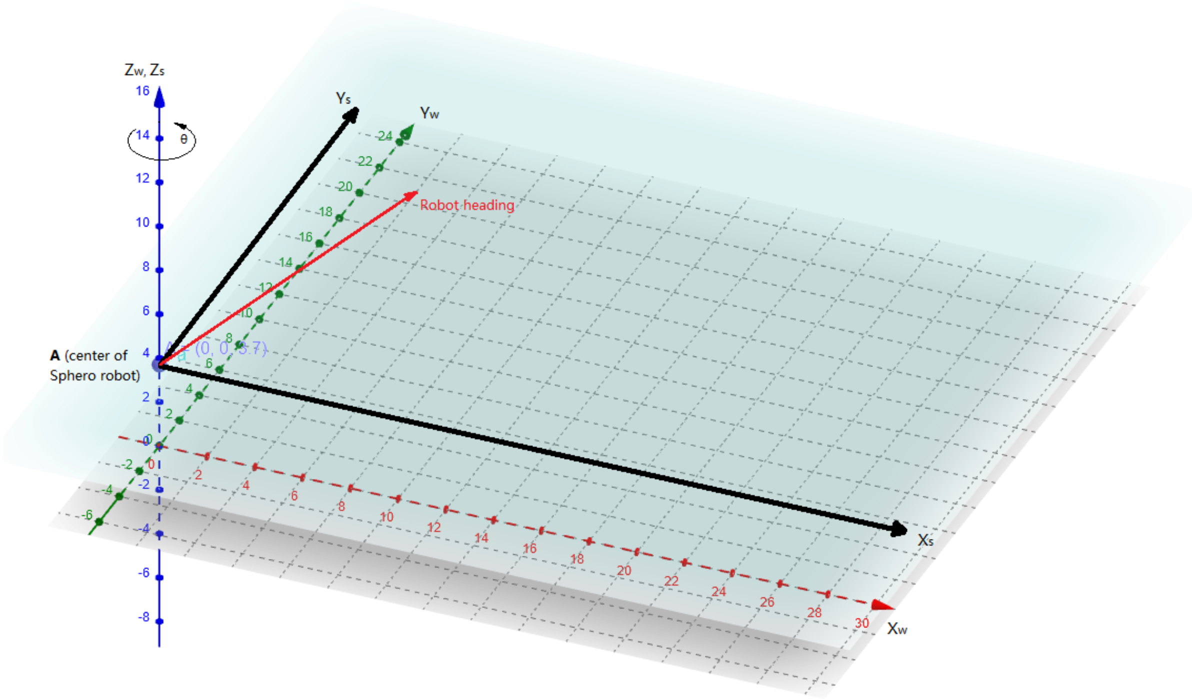

Figure 4.4: Reference flames in the Sphero robot

4.2.2 Calibration

The robot should be calibrated every time before staring a new exploration. The calibration aims at setting the initial position and the orientation to the Sphero robot. As we only work in a 2-dimensional environment, the current position of Sphero robot is the coordinates of the point where the shell is touching the ground or any other surface in the reference frame, the robot heading is the 0°. We show the reference frame in figure 4.4. Once it calibrated, Sphero's current position is $(0, 0)$ and the heading is aligned with the positive direction of the coordinate Y axis. Calibration sets the frame of the 2-D surface equal to the local frame of Sphero. The position and rotation are accumulated from the calibration.

We calibrate the robot by checking its backlight LED, the LED light is pointing at 180° to the robot heading. Because of lacking the accurate measurement, the calibration error is not always the same, which increases the system uncertainty. Calibration error mainly exists in two fields, position error and the orientation error. We have the position error because we are not sure if we exactly put the point Sphero touching the ground at the position we want. The orientation error may caused by the wrong placement and also the internal imbalance of the mobile robot. The calibration error will represented by a small error in the initial position.

4.2.3 Sphero Movement

Sphero could rotate to any direction in 2-D. As we explained in the hardware part of Sphero, the robot consists of two parts, one part is the inner mobile robot, another part is the outside plastic shell. The movement of sphero is mainly based on the friction. When the robot wants to move, the signal is sent the two motors, then the upper and lower rollers rotate at the same time, the friction between the shell and the wheel force the shell to rotate with it. At the same time, the friction between the shell and the ground causes the robot to roll.
The speed of the robot could be set from -255.0 to 255.0, this range translates to a maximum speed of $2m/s$, where forward is positive speed, backward is negative speed and when it stops, the speed is 0. The real-time speed value is also available by encoders on the motors through bluetooth.
The odometry data is from the integration of the accelerometers readings. It is not accurate. In particular, it ignores a situation that the Sphero slides along a obstacle, which is the main method we need to use in this project. In this project we used the 'wall following' strategy: the robot always follows a wall when exploring. Once it is in the corner, it will start a new exploration in another direction. When rolling along a wall, the robot cannot move according to the command, as shown in figure 4.5. When a command is sent to roll in direction A, the actual motion will be in the B direction (along to the wall). The angle between direction A and direction B is the exploration angle Θ. This discrepancy between the untended motion and the real one invalidates the odometry computed by the Sphero. Therefore we implemented our own odometry. For example, our exploration angle is $\Theta = 30°$, in the world frame, the vector $\vec{d} = (3.464, 2)$ is the odometry from the IMU board. We calculate the displacements δx and δy in both X and Y direction taken in to account the exploration angle Θ,

$$\delta x = \|\vec{d}\| \cdot sin(\Theta) \qquad \delta y = \|\vec{d}\| \cdot cos(\Theta) \tag{4.1}$$

Where $\|\vec{d}\|$ is computed by $\|\vec{d}\| = \sqrt[2]{3.464^2 + 2^2} = 3.99$. Because δx is much smaller than δy, so the odometry is $[0, \delta y]$ since walls are assigned to be either horizontal or vertical.

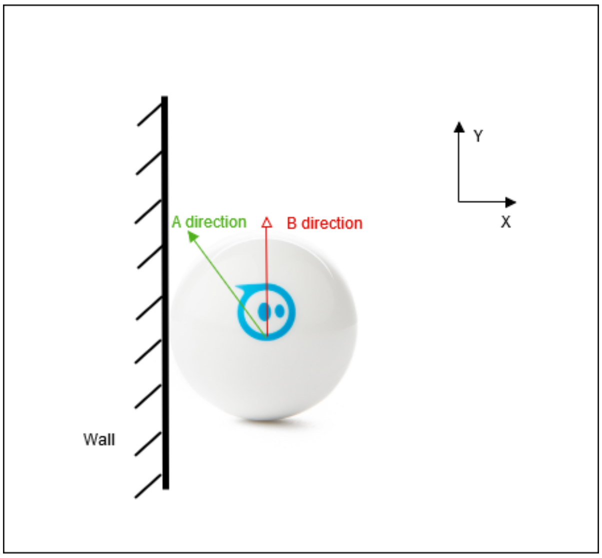

Figure 4.5: Sphero movement along the wall

Chapter 5

Experiments

In this chapter, we present the experimental evaluation of the approaches described in the previous chapters. In Section 5.1 the wall-following strategy and the experimental environment are introduced. In Section 5.2, the results of all approaches are shown and discussed.

5.1 Wall-following strategy

The wall-following strategy is based on 'left-hand rule' method. This method always follows the wall in the left side. If the robot meets a corner, rotates to continue following a wall [2]
The method to distinguish the convex and concave corners is provided next. At first, we wanted to use the collision detection to tell the difference between the corners. The build in collision is obtained easily through the IMU board sensor.
But when we tried in the experiment, it took us extremely long time to tune the sensitivity of collision detection due to the sensor's noise. So, we implement the corner detection based on speed change. The exploration should maintains the speed $50cm/sec$ along the direction A as in Figure 4.5. The actual speed we could get from the sensor (along direction B) would be lower the actual speed is $v_B = 50 \cdot \alpha$, where α is the cosine of the angle between direction A and B.
The main idea of this method is we keep tracking the value of velocity v_s, when abnormal velocity occurs, we consider an event (corner) comes. In this situation, only two types of corners are under our assumptions. ($\pm 90°$ corners)
When we consistently detect a reduction of velocity, we are in a convex corner. If the velocity is increase, it is a concave corner.

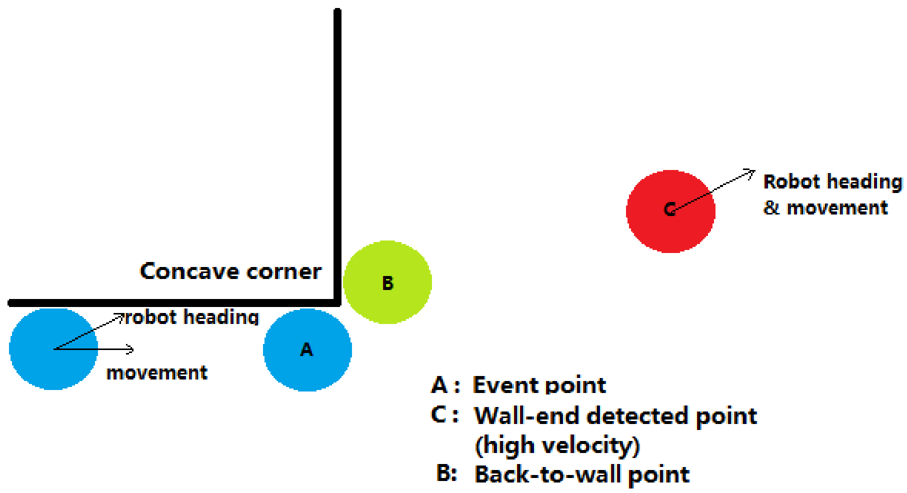

Figure 5.1: Concave corner solution.

When an unexpected increment in speed is detected. We stop the sphero, but as shown in Fig. 5.1, the Sphero robot (blue circle) at point C because of the sensor frequency and actuator delay. It will be very inaccurate if we consider the corner position at point C. Instead, we use one sample previous to the detection of the corner.
After the concave corner is detected, the robot rotates 177°and moves forward until a wall is detected, in order to reach the wall near the corner (point B in Fig 5.1). Therefore, the robot trajectory when it meets a concave corner is $A \to C \to B$ and the angle between vectors $\vec{AC}$ and $\vec{CB}$ is 177°. We assume that the obstacles in the environment are large enough so that this strategy always successfully overcomes convex corners.

In the experiment, we generate the trajectory by using the wall following strategy, save all the corner datas (time, corner type, coordinates from the odometer). Then we use Kalman filter approach, the ellipsoidal approach and the bounding box approach to process the dataset. Those procedure could be simply described as the flowchart 5.2,

5.1.1 Experimental Environment

Figure 5.3 and 5.4 show respectively a simple map environment and a complex map environment for our Sphero experiment.
The simple experiment is a rectangle,$X = 71.3cm$, $y = 46.3cm$ with 4 convex corners. The

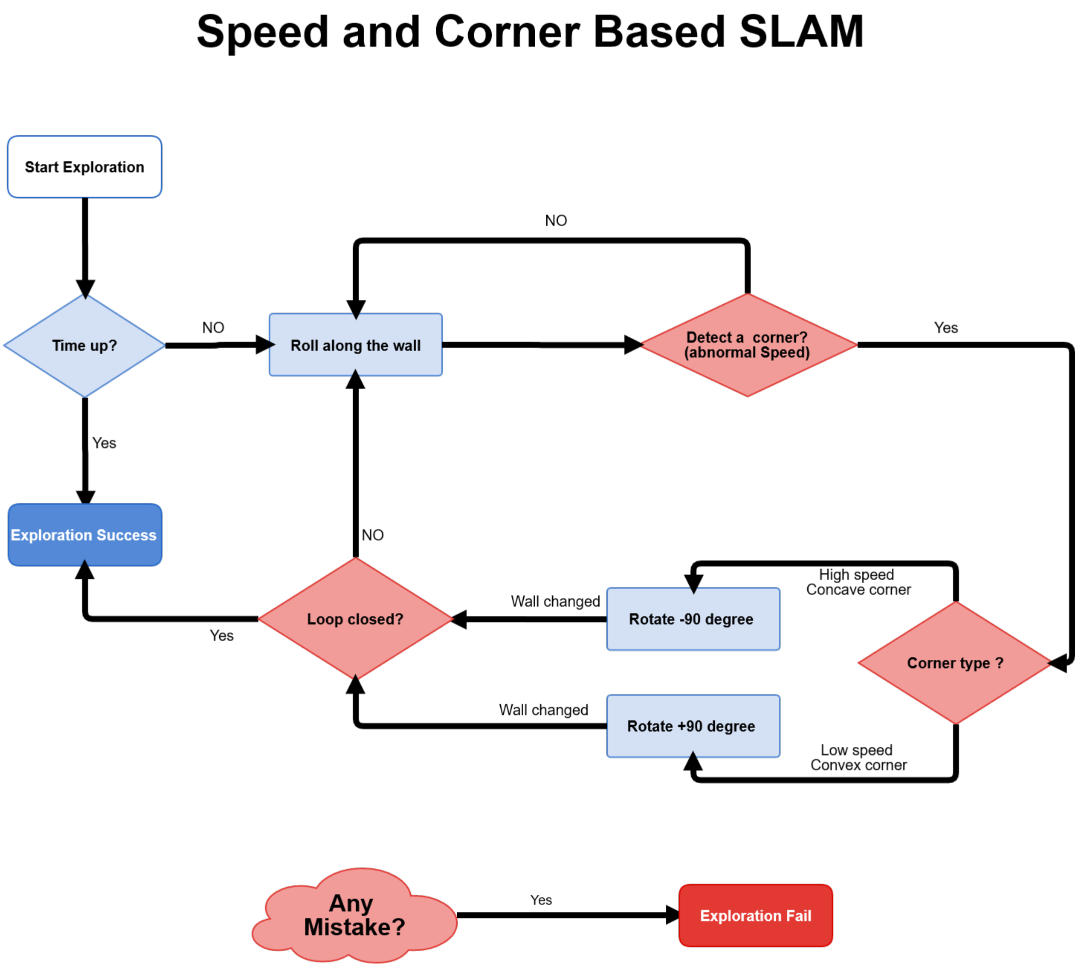

Figure 5.2: Collision and corner based wall-following strategy.

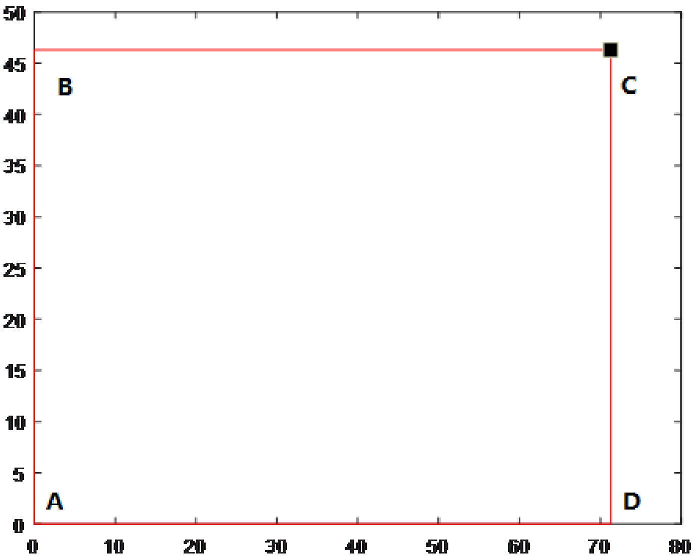

Figure 5.3: Simple map environment.

robot will start at point A, follow the path $A \rightarrow B \rightarrow C \rightarrow D$ then go back to point A.

Figure 5.4 shows the complex map environment. This test case includes a pair of each type of corners. The robots also traverses the environment clockwise. Figure 5.5 and figure 5.6 show the raw sensor data we get from the Sphero robot by applying our wall-following algorithm.

5.2 Experimental results

In the experimental results, we first shown the results of probabilistic approach and two set theoretic approaches before and after closing the loop in the simple environment. Then, we show the results for the complex environment with same order.

5.2.1 The simple environment

Figures 5.7, 5.8 and 5.9 show the results obtained with 3 methods presented in this thesis in the simple environment. Each figure shows the estimation before and after closing the loop.

From the results we could see that for the open loop, we have similar result with the three methods. Although the shape of the uncertainty is different, the trend of increasing is the same. The uncertainty become large and the uncertainty include the real corner. But we can see that in the ellipsoidal approach the uncertainty estimation grows faster. In other words, it

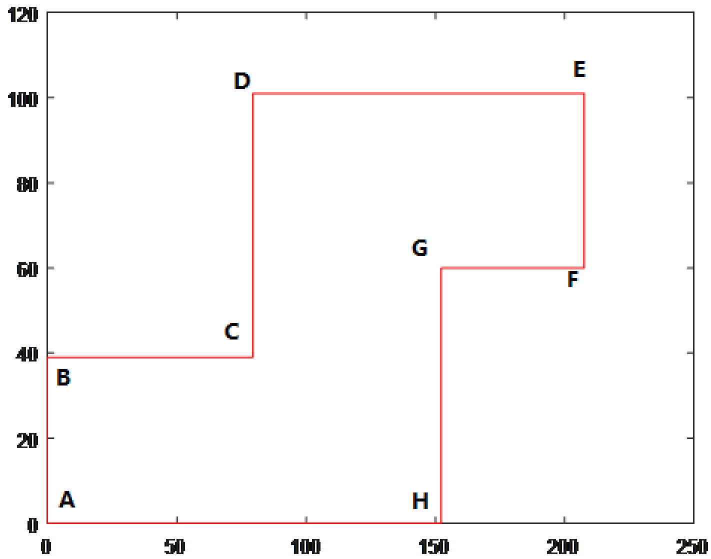

Figure 5.4: Complex map environment.

is less accurate, which means that if the loop is not closed, the robot could be in anywhere in the map.

In addition, as described in Chapter 3, we do data association calculation the ratio of intersection between the two ellipsoids. But in this case, if the last ellipsoid contains all the rest, the value W will be 1 for all the corners, i.e, the corner will be associated to the current one. Then we won't know where is the robot. To address this issue, the value W is considered not only the ratio of intersection between the two ellipsoids, but also the distance between the two centers of the ellipsoid.

$$DisR = 0.5 \cdot \max\left(\frac{1}{\|x_i - x_n\| + \|y_i - y_n\| + \delta}, 1\right) \tag{5.1}$$

$$W = 0.5 \cdot \max\left(\frac{|S_3|}{|S_1|}, \frac{|S_3|}{|S_2|}\right) + DisR = \frac{|S_3|}{min(|S_1|, |S_3|)} + DisR \tag{5.2}$$

Here $DisR$ is the distance ratio between the two ellipsoids, they are closer if the value is bigger. We put a constraint that the distance ratio is small or equals to 0.5, which means if the distance is less than 0.02cm, we consider the two center location has maximum likelyhood. δ is a small value close to 0 to be sure the denominator is not going to be 0 (current robot position is not exactly the same as one landmark visited before). In our experiment, we use $\delta = 0.0001$.

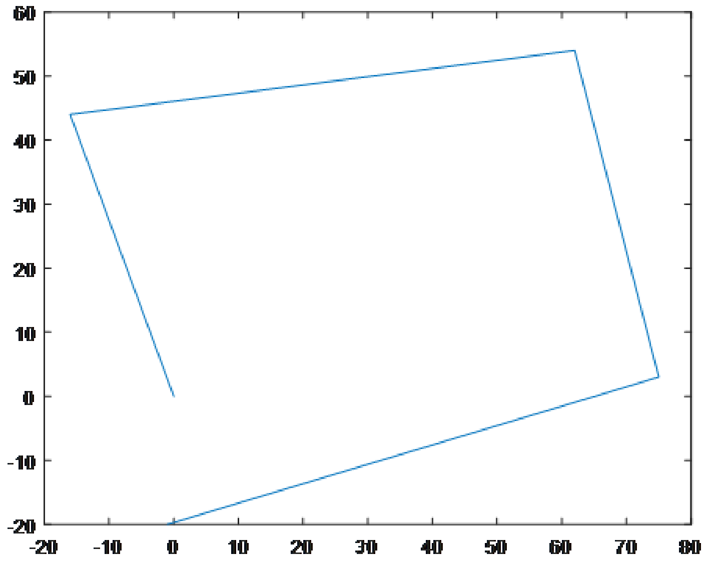

Figure 5.5: Raw sensor data in simple environment

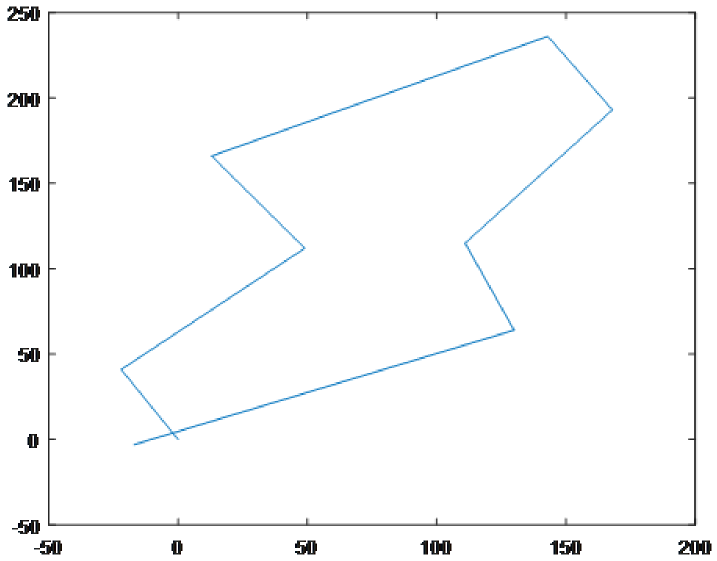

Figure 5.6: Raw sensor data in complex environment

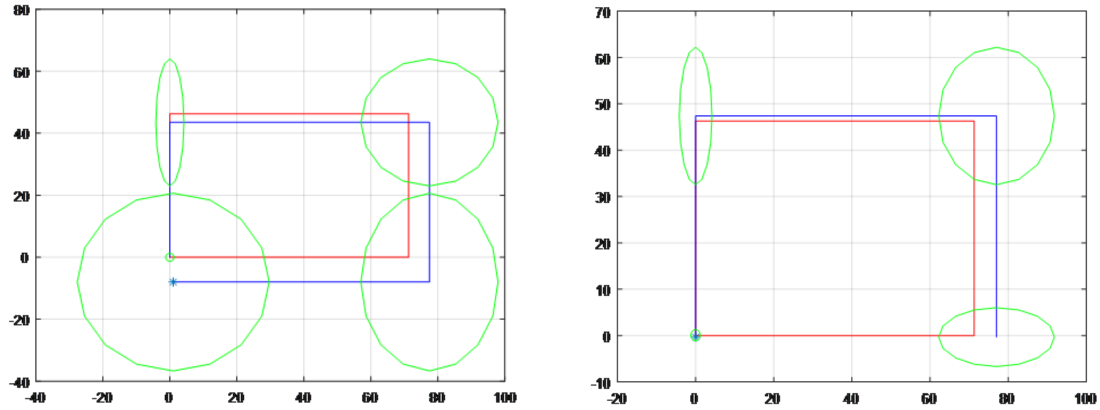

Figure 5.7: KF approach in simple environment.
Left: Before closing the loop. Right: After closing the loop.

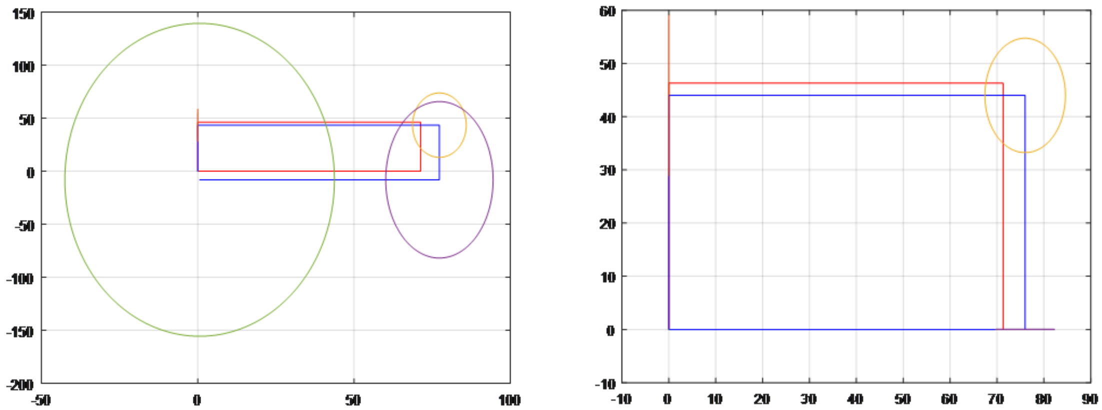

Figure 5.8: ellipsoidal approach in simple environment.
Left: Before closing the loop. Right: After closing the loop.

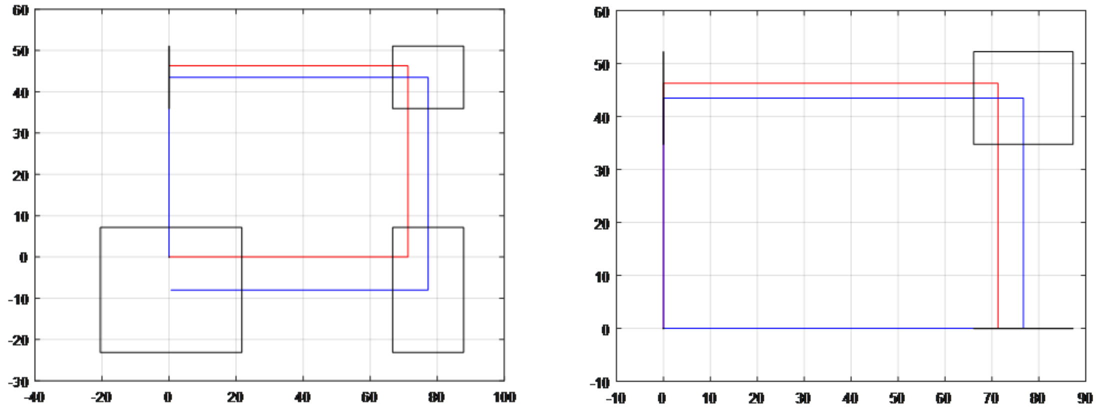

Figure 5.9: Bounding Box approach in simple environment.
Left: Before closing the loop. Right: After closing the loop

Here the value of the maximum similarity is still 1. The score W is between $[0-1]$, $W=0$ means there is no similar corner, $W=1$ means the current corner is 100% same as a one the robot been before.

Figure 5.10 shows the error size of different approach in simple environment, the error is calculated by,

$$e = \sum_{i=1}^{t} e_i \tag{5.3}$$

Where e_i is the error at corner $'i'$. For Kalman filter approach e_i is the volume of the ellipsoid at 99% confidence, for the set theoretic approaches is the volume of the set bounded the corresponding error. From the results, we could conclude:

1. All the methods are correct, the real corner's position are all included in the uncertainty ranges both before and after closing the loop.
2. For all the methods, the uncertainty is reduced when closing the loop, especially for the loop closure corner and the ones close to it.
3. For all the methods, the loop closure corner has the lowest error. The error increase in both sides of the trajectory, forwards and backwards from the loop closure corner.
4. For the Kalman filter approach, the error in loop closure corner is not decreased to the initial uncertainty. The uncertainty is in between the two estimations, since it is a mean with different weights.

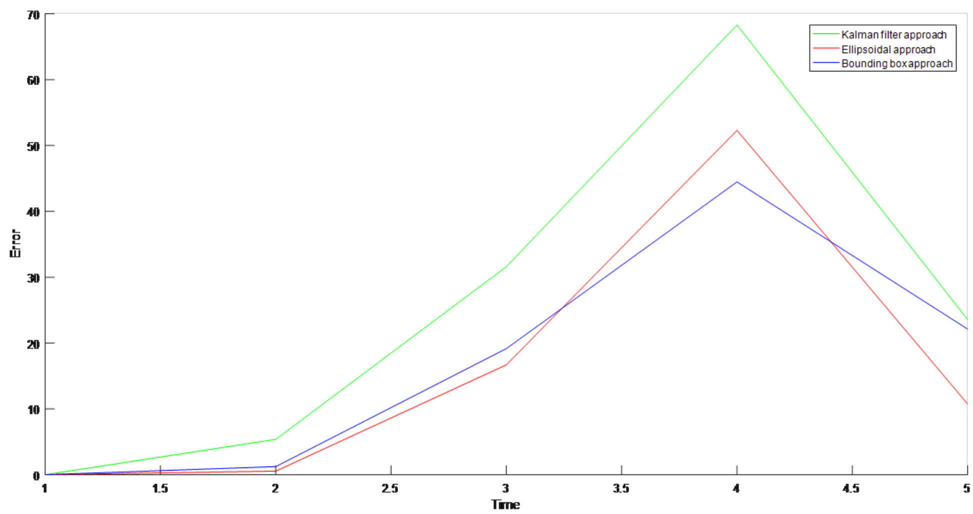

Figure 5.10: Error size of three approach in simple environment

5. For the set theoretic approach, the error in the estimation of the loop closure corner decreased to exactly the size of initial uncertainty. In our experiments, the ellipsoid approach and the bounding box approach yield more or less the same results, because we consider the common part of the current uncertainty is the fusion of the two ellipsoids/bounding box. In addition, the ellipsoid representing the initial uncertainty in the first corner is included in the ellipsoid representing the uncertainty at the end of the trajectory, the fusion of the two ellipsoid is the initial ellipsoid.

5.2.2 The complex environment

Figures 5.11, 5.12 and 5.13 show the results obtained with 3 methods presented in this thesis in the complex environment. Each figure shows the estimation before and after closing the loop. Figure 5.14 shows the error (see Eq. 5.3) of the different approaches in the complex environment. For Kalman filter approach, e_t is the volume of the ellipsoid at 99% of confidence.

In the complex environment, we observe results similar to those in the simple environment. The uncertainty is accurate enough to include the real corner's position.

Also, the set theoretic approach is better because it reduces more the uncertainty. The bounding box approach is simpler than the ellipsoidal approach, but it is constrained to a linear model. If the model is more complex, it would be probably better to use the ellipsoidal approach.

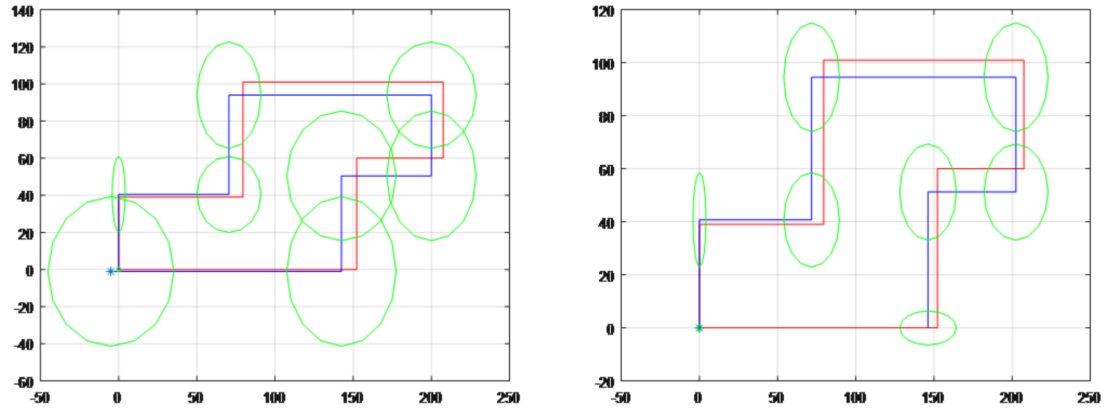

Figure 5.11: KF approach in complex environment.
Left: Before closing the loop. Right: After closing the loop.

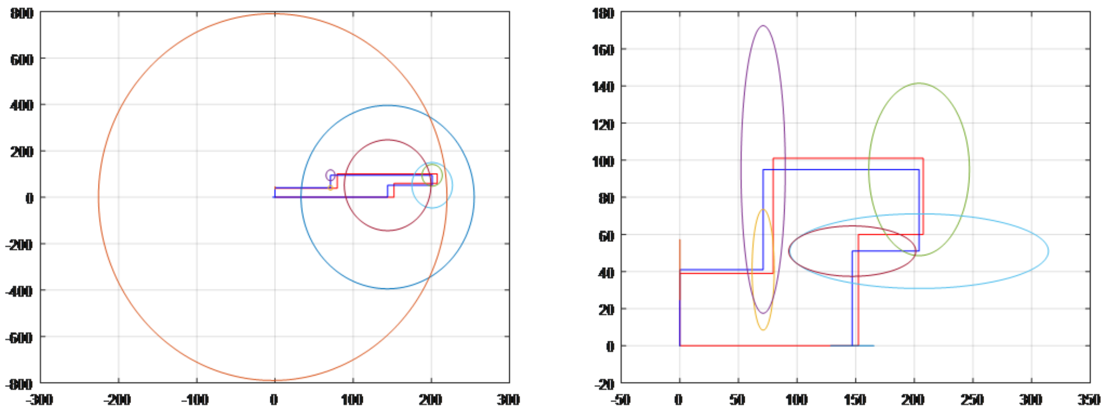

Figure 5.12: Ellipsoidal approach in complex environment.
Left: Before closing the loop. Right: After closing the loop.

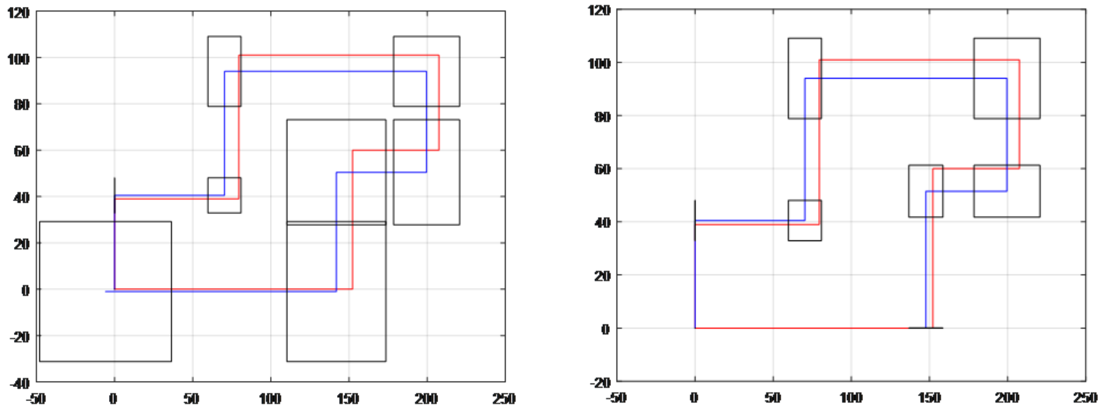

Figure 5.13: Bounding Box approach in complex environment.
Left: Before closing the loop. Right: After closing the loop.

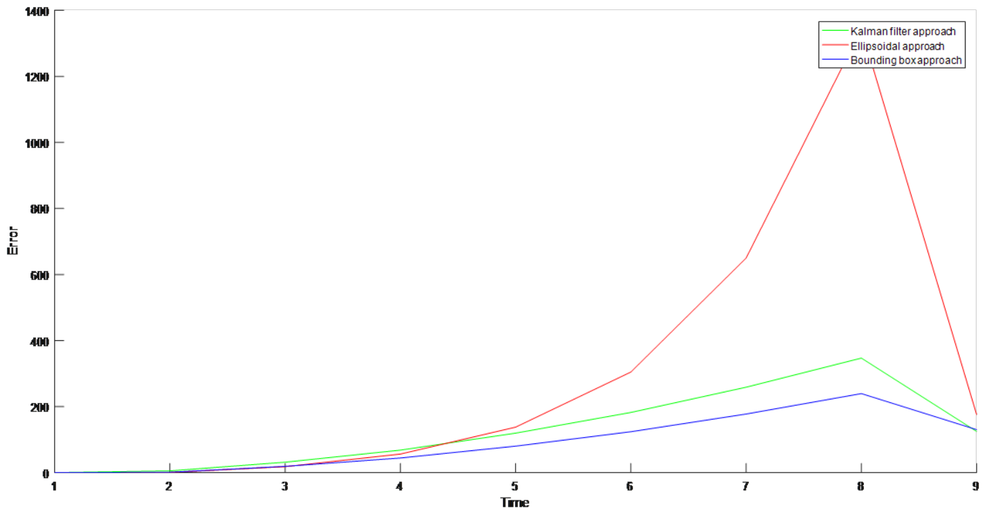

Figure 5.14: Error size of three approach in complex environment

Chapter 6

Summary and Future Work

In this thesis, a corner-based wall following strategy is presented and it is used to solve the SLAM problem in environment with axis aligned walls. Most of the efforts in this thesis focused on the low level aspects such as the access to the Sphero sensors and controls, the implementation of a wall following strategy, or the detection of corners on a reliable way. However this thesis also contributes implementing SLAM strategies based on these different ways to represent the uncertainly in the problem.

In the experiment, we use Matlab to test and compare the results of Kalman filter approach and two types of set theoretic approaches running on the same environments. From the results, the set theoretic approaches could be considered an efficient way to solve the SLAM problem because they generate better estimations after closing the loop. Nevertheless, the environments considered in this thesis are simple, e.g., the motion and observation models are linear. Different results may be obtained in more complex setting with non linear models.

This thesis can be extended in several directions:

1. Better hardware and complex model representation.
   The current research is constrained in several ways. However in real life, we need to consider more possibilities. The path may not limited to rectilinear environment, the ground may not smooth and sometimes we cannot avoid small obstacles in the path which may lead to system instability. Some of these problems could be analyzed if we have better sensor. Also, a more complex model to represent the robot movements and observations would be necessary.

2. Other types of uncertainty description.
   In the thesis, two different ways of modeling the uncertainty are studied in a set-theoretic

point of view. The the approach is favored due to the non-Gaussian robot pose distribution. It would be interesting to implement alternative ways to describe uncertainty and the corresponding propagation and fusion methods.

3. Multiple robot SLAM
In this thesis, only one robot is used in the experiments. It will be interesting to have multiple robot to reduce the exploration time. For this it would be necessary to develop tools to fuse the information obtained by each robot.

Appendix

Kalman filter approach

main.m

```
%% main run in sphero
% This function is the main function for recording datas
% Also the Kalman filter approach result is test directly in this experiment
% map and P are the states and the covariance after loop-closing
% map1 and P1 are the states and the covariance before loop-closing
% After run this file, 'xlog','ylog','t' and 'cornerlog' need to be saved ...
    for other approaches.

%% connect sphero robot and set up the settings
    sph = connectsph(); % call function 'connectsph' to establish the ...
        connection between Matlab and Sphero robot.
    setsystem(sph);     % call function 'setsystem' to set systematic ...
        parameters.

%% parameter declaration

    idx=1;    % the index i-th of wall the robot following
    label=0;  % lable is to declare if the new state is part of mapped states
    loop = 1; % times of loops --> for balancing the time for recording the ...
        data and the motion timeout

% parameter for corners
    punishment = 0;  % for detecting the low speed, corner type 1
    signal_wall = 0; % for detecting the high speed, corner type 2
    corner_type = 0; % corner type signal:  corner_type = 0 --> no corner ...
        detected
```

```
            %               corner_type = 1 --> corner type 1 detected
            %               corner_type = 2 --> corner type 2 detected
    cornerlog = [];  % List of all the corner type from the initial robot ...
        position

% Sensor paramters initialization:
    % After calibration(in function'setsystem'), initialize the sensor ...
        parameters.
    [xstart, ystart, ¬, ¬, groundspeed] = readLocator(sph);
    [accX, accY, accZ] = readSensor(sph, {'accelX', 'accelY', 'accelZ'});
    % temporary parameters, because the values from sensor are integer.
    xcur = double(xstart); % initial robot position x
    ycur = double(ystart); % initial robot position y
    senpx = 0;  % position x from sensor
    senpy = 0;  % position y from sensor
    senvx = 0;  % velocity vx from sensor
    senvy = 0;  % velocity vy from sensor

% recording parameters
    xlog = xcur;
    ylog = ycur;

% Parameters for SLAM
    % Map after closing the loop
    P = [0.1 0;0 0.1]; %initial covariance ;
    map = [xcur;ycur];  % map % initial map (allhe states) [Robot x ;robot ...
        y;Landmark1 x ;y;L2 x;y]
    % MAP before closing the loop, for the comparation
    P1 = [0.1 0;0 0.1];
    map1 = [0;0];

% parameters for the system
    tfinal = 90; % Time limit on the motion of the Sphero
    espeed = 60; % exploration speed
    angle = -25; % initial angle for exploration

% timers
t = 0;
t0 = cputime;
tt1 = t0;

```

```
%% the main loop for exploration
    tic % start timer
while (toc < tfinal) % main loop

    % movement command, towards initial heading 'angle' direction, with ...
        initial exploration speed
    result = roll(sph, espeed, angle);

    % 9s per each small loop because we set the motion timeout as 10s
    while ( signal_wall==0   &&  toc < loop*9  && toc < tfinal)

        % if events not happend, keep recording the pos,velocity
        % save the locator in the fastest speed in order to detect if wall ends
        [x,y,vx,vy,¬] = readLocator(sph);
        senpx(end+1) = double(x);
        senpy(end+1) = double(y);
        senvx(end+1) = double(vx);
        senvy(end+1) = double(vy);

        %  Corner type detection  1) Continuesly Low velocity: in the ...
            corner type 1
        %                        2) The Velocity increase instantaneously: ...
            in the corner type 2

        %%  corner type 1
        if  (length(senvx)>2) && (abs(senvx(end)−senvx(end−1)) ...
                                +abs(senvy(end)−senvy(end−1)) < 5 )
            punishment = punishment+1;
            if  punishment ≥ 5  % if continuesly Low velocity
                corner_type = 1;
                [map,P,map1,P1,t,angle,xlog,ylog] = ...
                    newevent(sph,angle,map,map1,P,P1,xlog,ylog,t,tt1,idx);
                % if we detect a corner, we reset the angle and break this loop
                % rotate 90 degree to continue exploration
        [angle,corner_type,idx,cornerlog] = ...
            angle_change(angle,corner_type,idx,cornerlog);
                break
            end
        end

        %%  wall ends ?  (corner type2)
```

```
        % if the velocity increase instantaneously --> signal_wall = 1, or ...
            else the signal will remain 0.
        if  (length(senpx)>2)
            direction = getdirection(senpx,senpy,angle);
        % tracking the position,velocity and check if the velocity change ...
            instantly.
            [senpx,senpy,senvx,senvy,signal_wall] = ...
                check_if_wall_ends(sph,senpx,senpy,senvx,senvy,direction);
        end
        if signal_wall==1
            % if wall ends detect, go back to the wall
            signal_wall = back_to_wall(sph,angle);
            % the new event is in the remembered position
            [map,P,map1,P1,t,angle,xlog,ylog]= ...
            newevent(sph,angle,map,map1,P,P1,xlog,ylog,t,tt1,idx);
            corner_type=2;
        % rotate -90 degree to continue exploration
            [angle,¬,idx,cornerlog] = ...
            angle_change(angle,corner_type,idx,cornerlog);
            break
        end
    end
    % reset punishment and increase the loop size
    punishment = 0;
    loop = loop+1;
end
brake(sph); % when the exploration succeed, release sphero obeject.

%% plot
plotfigure(map,P);  % plot after closing the loop
figure,plotfigure(map1,P1);  % plot before closing the loop
```

mainsim.m

```
%% main run simulation with record datas
% This function is the main function for kalman filter approach
% map and P are the states and the covariance after loop-closing
% map1 and P1 are the states and the covariance before loop-closing

% initialize the map and covariance
    map = [0;0];  % map [Robot x ; robot y; Landmark1 x ; y; Landmark2 x; y ...
```

```
            ... Landmarkn x; y]
    P = [0.1 0;0 0.1]; %initial covariance ;
    % for the comparation(before loop-closing)
    map1 = [0;0];
    P1 = [0.1 0;0 0.1];

% load('**.mat'); % load data

% Parameters for looping
    xlog = x(1);
    ylog = y(1);
    angle = -25; % initial angle for exploration

% Main loop
    for i = 2:length(x)
        delt = t(i)-t(i-1); % Δ t
        xlog = [xlog,x(i)]; % sensor position x
        ylog = [ylog,y(i)]; % sensor position y
    % Get estimate change of P, R robot, Prr, estx, esty
        [P,map] = getchange(angle,map,xlog,ylog,delt,P);
        [P1,map1] = getchange(angle,map1,xlog,ylog,delt,P1);
        if i<3
             % add statas(the first landmark) to map
             [map,P] = addtostate(map,P,i-1);
             map1 = map;
             P1 = P;
        else
             % Pridiction, form the P covariance.
             [map,P] = prediction(map,P);
             [map1,P1] = prediction(map1,P1);
             % see if position now is part of the mapped landmark
             [label,corl] = ifnewstate(map,P);
             if label==0  % never seen this state
                  [map,P] = addtostate(map,P,i-1);
                  % compared map
                  [map1,P1] = addtostate(map1,P1,i-1);
             else
                  [map,P]=correctmap(map,P,corl);
                  % compared map
                  [map1,P1]=addtostate(map1,P1,i-1);
             end
```

```
            end
            % rotate for next wall fllowing
            corner_type=cornerlog(i−1);
            [angle,¬,idx,cornerlog] = ...
                    angle_change(angle,corner_type,i−1,cornerlog);
            angle = mod(angle, 360);
        end
    % plot
        figure,plotfigure(map1,P1); % plot before closing the loop
        figure,plotfigure(map,P); % plot after closing the loop
```

connectsph.m

```
%% Create a Sphero object (if it does not exist)
% Connect Matlab with Sphero robot
% % This function only used in the real robot exploration

% Output
%       sph        −  Sphero robot object

function sph=connectsph()
    % if the object doesn't exist, Create a Sphero object
    if ¬exist('sph','var')
        sph = sphero();
    end

    % Set up connection
    connect(sph);

    % ping
    result = ping(sph);

    % interrupt the example if ping was not successful
    if ¬result
        disp('Example aborted due to unsuccessful ping');
    return,
    end
end
```

setsystem.m

```
%% This function is for setting sphero robot

function setsystem(sph)
    % set every rotation time last 10 seconds
    sph.MotionTimeout=10;

    % Turn on handshaking
    sph.Handshake = 1;

    % turn on the back LED
    % Easily check the heading of Sphero robot
    sph.BackLEDBrightness = 255;

    % Calibrate the orientation of the sphero.
    % initialize the orientation of the Sphero in the desired direction.
    calibrate(sph, 0);

    % turn on the collision detection
    sph.CollisionDetection=1;
end
```

getchange.m

```
% This function is to get the new robot position and related covariance
% It's part of prediction in Kalman filter approach

% Input:
%    angle    - angle beteween robot heading and initial wall (positive in ...
    clockwise)
%    map - Map (states, include the current robot position)
%    xlog     - List of landmark position x
%    ylog     - List of landmark position y
%    P        - covariance (include the current robot covariance)

% Output:
%    map     - updated Map
%    P       - updated covariance


function [P,map] = getchange(angle,map,xlog,ylog,delt,P)
```

```
    % parameters for error
    varp = 0.005;
    % F according to the state change
    func=eye(2);
    jFr = eye(2);
    jFn = eye(2);
    % calculate the measurment data according to x and y aixes :
    rtheta = deg2rad(abs(angle));  % rtheta is the rubot running heading ...
        angle in world frame,rad
    if  abs(sin(rtheta))*abs(xlog(end)−xlog(end−1))≥ ...
                abs(cos(rtheta))*abs(ylog(end)−ylog(end−1))
        if  rtheta < pi % x increasing
            % measurment accumulated
            senx =  + abs(xlog(end)−xlog(end−1));
            seny = 0;
        else  % x decreasing
            % measurment accumulated
            senx = − abs(xlog(end)−xlog(end−1));
            seny = 0;
        end
        q = varp*delt+0.5;
        Q = [50*q,0;0 0];
    else
        if  rtheta < pi/2 || rtheta > 3*pi/2 % y increasing
            % measurment accumulated
            senx = 0;
            seny = + abs(ylog(end)−ylog(end−1));
        else
            senx = 0;
            seny = − abs(ylog(end)−ylog(end−1));
        end
        q = varp*delt+0.5;
        Q = [0 0;0 50*q];
    end
    map(1:2,1)  = func*map(1:2,1)+[senx;seny];
    P(1:2,1:2)  = jFr*P(1:2,1:2)*jFr'+jFn*Q*jFn';
end
```

ifnewstate.m

```
% see if the current position is associated to one mapped landmark
```

```
% Input:
%   map      — Map (states, include the current robot position)
%   P        — covariance (include the current robot covariance)
% Output:
%   label    — signal for data association   label = 0 no associated ...
    landmark(new corner);
%                          label = 1 landmark associated
%   corl     — pointer of associated landmark (—th) in the state

function [label,corl] = ifnewstate(map,P)

    % initial min distance of mahanobian distance
    mindis=1e+06;  % a initial value (big)
    % total numbers of the visit corner
    n=(length(map)/2);
    % corl—— landmark label
    corl=0;

    % find minimum M distance between current robot position to every ...
        previous landmark
    for i=2:n
        dif = [map(1,1);map(2,1)]—[map(2*i—1,1);map(2*i,1)];
        Pdif = P(1:2,2*i—1:2*i);
        % manhanobian distance
        madis = abs(dif'*Pdif*dif);
        if madis < mindis
        mindis = madis;
        corl = i;   % cor = 2 ——> corner 0
        end
    end


    % we compare the min distance to a threthold
    if mindis<300
        % if its small enough
        label = 1;
    else
        label = 0;
    end

end
```

prediction.m

```
% obtain the current estimate position of the new state

% Input:
%   map — Map (states, include the current robot position)
%   P   — covariance (include the current robot covariance)
% Output:
%   map — updated Map
%   P   — updated covariance

function [map,P]=prediction(map,P)

    jFr = eye(2);
    % robot position prediction
    % alread done in getchage

    % covariance prediction
    Prm = jFr*P(1:2,3:end);
    P(1:2,3:end) = Prm;
    P(3:end,1:2) = Prm';
end
```

angle change.m

```
%% This function is called when the corner detect.
% it aims at changing the robot heading in order to follow another wall

% Input:  angle     —  angle beteween robot heading and initial wall ...
    (positive in clockwise)
%     corner_type    —  corner type signal
%     idx       —  pointer of landmark (—th wall)
%     cornerlog —  List of all the corner type from the initial robot position

% Output: angle     —  angle beteween robot heading and initial wall ...
    (positive in clockwise)
%     corner_type    —  updated corner type signal
%     idx       —  updated pointer of landmark (—th wall)
%     cornerlog —  List of all the corner type from the initial robot position

```

```

function [angle,corner_type,idx,cornerlog]=angle_change ...
            (angle,corner_type,idx,cornerlog)

    % change angle, when next command send to robot, the robot follows ...
        another wall.
    if corner_type ==1   % convex corner
        angle = angle + 90;
    else
        if corner_type == 2   % concave corner
            angle = angle - 90;
        end
    end

    % add the current corner type in the list
    cornerlog=[cornerlog,corner_type];
    corner_type = 0; % reset corner_type

    % set angle to the range ( 0-360 )
    angle = mod(angle, 360);

    % increase the index
    idx = idx+1;

end
```

check if wall ends.m

```
%% This function checks if there is a wall end with maximum frequency.
% In experiment, the frequency is 2Hz
% This function only used in the real robot exploration

% Input:
%     sph       -  Sphero robot object
%     senpx     -  List of robot position x from sensor
%     senpy     -  List of robot position y from sensor
%     senvx     -  List of robot velocity vx from sensor
%     senvy     -  List of robot velocity vy from sensor
%     direction -  robot heading direction

% Output:
```

```
%     senpx      —  List of robot position x from sensor
%     senpy      —  List of robot position y from sensor
%     senvx      —  List of robot velocity vx from sensor
%     senvy      —  List of robot velocity vy from sensor
%     signal_wall    —  signal for if the wall ends  (= 0 wall end not ...
    detected; = 1 detected wall end)

function    [senpx,senpy,senvx,senvy,signal_wall] = ...
    check_if_wall_ends(sph,senpx,senpy,senvx,senvy,direction)
    signal_wall = 0;  % set the siganl remain 0 (wall not end);

    % keep recording
    [xcur, ycur,vx,vy,groundspeed] = readLocator(sph);
    senpx(end+1) = double(xcur);
    senpy(end+1) = double(ycur);
    senvx(end+1) = double(vx);
    senvy(end+1) = double(vy);


    % we consider wall end if satisfy those conditions:
    if direction == 1 || direction == 2 % along the x
        if ((abs(vy)>300) && (abs(senvy(end)—senvy(end—1))>150) && ...
            abs(groundspeed>480))||(abs(groundspeed)>600)
            signal_wall=1;
        end
    else
        if direction == 3 || direction == 4 % along the y positive
            if ((abs(vx)>300) && (abs(senvx(end)—senvx(end—1))>150)&& ...
                abs(groundspeed>480))||(abs(groundspeed)>600)
                signal_wall=1;
            end
        end
    end

end
```

addtostate.m

```
%% This function is for landmark initialization
% For adding the corner in the states

```

```
% Input:
%   map — Map (states, include the current robot position)
%   P   — covariance (include the current robot covariance)
%   idx — pointer of landmark (—th) needs to add in Map
% Output:
%   map — updated Map
%   P   — updated covariance

%%
function [map,P]=addtostate(map,P,idx)

%% jacobians
Gr = eye(2);
Gy1 = eye(2);
Q = eye(2);

     if idx<2
    % initialize the Covariance matrix and add the first landmark's covariance
        map=[map;0;0];
        P(end+1:end+2,1:end)=0.5*eye(2);
        P(1:end—2,end+1:end+2)=0.5*eye(2);
        P(end—1:end,end—1:end)=0.1*eye(2);

        Prr = P(1:2,1:2);
        Prm = P(1:2,3:end);
        Pll=Gr*Prr*Gr'+Gy1*Q*Gy1';
        Plx=Gr*[Prr Prm];

        P(end+1:end+2,1:end)=Plx;
        P(1:end—2,end+1:end+2)=Plx';
        P(end—1:end,end—1:end)=Pll;
     else
    % Add landmark to the covariance
        Prr = P(1:2,1:2);
        Prm = P(1:2,3:end);
        Pll=Gr*Prr*Gr'+Gy1*Q*Gy1';
        Plx=Gr*[Prr Prm];

        P(end+1:end+2,1:end)=Plx;
        P(1:end—2,end+1:end+2)=Plx';
        P(end—1:end,end—1:end)=Pll;
```

```
    end

map = [map;map(1,1);map(2,1)]; % renew Map
end
```

correctmap.m

```
%% This function is used after loop closure for correcting the Map and ...
    related covariance.

% Input:
%   map — Map (states, include the current robot position)
%   P   — covariance (include the current robot covariance)
%   corl— pointer of associated landmark (—th) in the state
% Output:
%   map — updated Map
%   P   — updated covariance

function [map,P]=correctmap(map,P,corl)
    Prr = P(1:2,1:2);
    Prl = P(1:2,(corl*2—1):(corl*2));
    Pll = P((corl*2—1):(corl*2),(corl*2—1):(corl*2));
    Pmr = P(3:end,1:2);
    Pml = P(3:end,(corl*2—1):(corl*2));
    % H jacobian
    jHr = eye(2);
    jHl = eye(2);

    R = 1.5*eye(2); % Noise of measurement
    z = [map(corl*2—1);map(corl*2)]—[map(1);map(2)];  % landmark — observation
    Z = [jHr jHl]*[Prr Prl;Prl' Pll]*[jHr';jHl']+ R;  % here plus the ...
        covariance of noise of measurement
    K = [Prr Prl; Pmr Pml]*[jHr';jHl']*(Z^(—1)); % Kalman gain
    kz = K*z;
    kzk = K*Z*K';

    % correct map
    map = map + kz;
    % correct covariance
    P = P — kzk;
end
```

getdirection.m

```
% This function is to get the new robot heading direction

% Input:
%   angle   — angle beteween robot heading and initial wall (positive in ...
    clockwise)
%   map — Map (states, include the current robot position)
%   xlog    — List of landmark position x
%   ylog    — List of landmark position y
%   P       — covariance (include the current robot covariance)

% Output:
%   map     — robot heading direction  (in world frame)
%               direction = 1 ——> robot heading at x positive
%               direction = 2 ——> robot heading at x negative
%               direction = 3 ——> robot heading at y positive
%               direction = 4 ——> robot heading at y negative

function direction = getdirection(senpx,senpy,angle)
% calculate the measurment data according to x and y aixes :
    rtheta = deg2rad(abs(angle));  % rtheta is the rubot running heading ...
        angle in world frame,rad
    if  abs(senpx(end)—senpx(end—1))>abs(senpy(end)—senpy(end—1))
        if  rtheta < pi % x increasing
            direction = 1;
        else  % x decreasing
            direction = 2;
        end
    else
        if  rtheta < pi/2 || rtheta > 3*pi/2 % y increasing
            direction = 3;
        else
            direction = 4;
        end
    end
end
```

newevent.m

```
%% After we detect an event happends(a corner detected),
% we need to check if this corner is a new one or a old one we already visited.
% Then deal with them with different ways.
% If it's a new corner --> add to states
% If it's a corner already visited --> close the loop
%
% Input parameters:
%           sph     -  Sphero robot object
%           angle   -  angle beteween robot heading and initial wall ...
    (positive in clockwise)
%           map     -  Map (states, include the current robot position)
%           map1    -  reference map (before closing the loop, for comparasion)
%           P       -  covariance (include the current robot covariance)
%           P1      -  reference covariance matrix (before closing the ...
    loop, for comparasion)
%           xlog    -  all landmarks x position
%           ylog    -  all landmarks y position
%           t       -  event occur time list
%           tt1     -  last event occur time (for computation Δ t)
%           idx     -  index of the wall following
%
% Output parameters:
%           map     -  updated Map
%           P       -  updated covariance matrix
%           map1    -  updated reference map
%           P1      -  updated reference covariance matrix
%           angle   -  input of sphero, robot heading
%           xlog    -  all landmarks x position
%           ylog    -  all landmarks y position
%           t       -  event occur time
%           tt1     -  last event occur time (for computation Δ t)
%           idx     -  index of the wall fllowed
function [map,P,map1,P1,t,angle,xlog,ylog]= ...
newevent(sph,angle,map,map1,P,P1,xlog,ylog,t,tt1,idx)
    % Read the current position and speed of the robot
    [xcur, ycur,¬, ¬, ¬] = readLocator(sph);
    brake(sph);

    t(end) = toc;  % event time
    delt=cputime-tt1;
    xlog(end+1) = double(xcur);
```

```
    ylog(end+1) = double(ycur);
        % then we calculate if it is the privious state, if it is, close the
        % loop , if its not,initialize the landmark and go for another run.

    % get estimate change of P,R robot,  Prr, estx, esty
    [P,map] = getchange(angle,map,xlog,ylog,delt,P);
    [P1,map1] = getchange(angle,map1,xlog,ylog,delt,P1);
    if idx<2
        % add statas(the first landmark) to map
        [map,P] = addtostate(map,P,idx);
        map1 = map;
        P1 = P;
    else
        % Pridiction, form the P covariance.
        [map,P] = prediction(map,P);
        [map1,P1] = prediction(map1,P1);
        % see if position now is part of the mapped landmark
        [label,corl] = ifnewstate(map,P);

        if label == 0  % never seen this state
            [map,P] = addtostate(map,P,idx);
            % compared map
            [map1,P1] = addtostate(map1,P1,idx);
        else
            [map,P] = correctmap(map,P,corl);
            % compared map
            [map1,P1] = addtostate(map1,P1,idx);
        end
    end
end
```

plotfigure.m

```
% This function is for plot the Map and Covariance

function plotfigure(map,P)


    % % real simple environment
    % M00 = [0   0   75  75   0
    %        0   50  50  0   0];
```

```
    % ball is 7.4 cm
    % simple environment without the ball length in 2-D frame
    M00 = [0     0     71.3   71.3    0
                0    46.3    46.3    0    0];

    % % big map without the ball length in 2-D frame
    %  M00 = [0     0    79.5      79.5    207.5     207.5  152.2    152.2     0
    %          0    39     39       101      101      60        60       0    0];


    % total numbers of the visit corner
    n=(length(map)/2);

    even = map(4:2:end,1);
    odd = map(3:2:end,1);

    plot(M00(1,:),M00(2,:),'r');  % red is the real map
    hold on
    grid on
    plot(odd,even,'b');  % blue is after the karman
    plot(map(1),map(2),'*');

    % plot the covariance
    for i=2:n
        xxx = [map(2*i-1,1);map(2*i,1)];
        ppp = P(2*i-1:2*i,2*i-1:2*i);
        % plot 3 sigma ellipse's coordinates
        [xx,yy] = plotllipse(xxx, ppp, 4);
        plot(xx,yy,'g');
    end

end
```

plotllipse.m

```
%% This function is for ploting the covariance for Kalman filter approach

function [X,Y] = plotllipse(x,P,n,NP)
    if nargin < 4
        NP = 16;
        if nargin < 3
```

```
                n = 1;
            end
        end
        alpha = 2*pi/NP*(0:NP); % NP angle intervals for one turn
        circle = [cos(alpha);sin(alpha)]; % the unit circle
        % SVD method, P = R*D*R' = R*d*d*R'
        [R,D]=svd(P);
        d = sqrt(D);
        ellip = n * R * d * circle;
        % output ready for plotting (X and Y are line vectors)
        X = x(1)+ellip(1,:);
        Y = x(2)+ellip(2,:);
```

Ellipsoidal approach

ee main.m

```
%% main run simulation with record datas
% This function is the main function for Ellipsoidal approach
% back_pos and back_P are the states and the covariance after loop−closing
% map_pos and mao_P are the states and the covariance before loop−closing

% initialization
label=0;

% robot
r_pos = [0;0] ;
r_P = [100 0;0 100]; %initial error;

change_pos = [];
change_P = [];
disp_pos = [0;0]; % displacment
disp_P = [100 0;0 100];

% map:
map_pos = [0;0];  % initial map with the current robot position
map_P = [100 0;0 100]; %initial error;

%load('cornerdatabigmap.mat');
```

```

xlog = x(1);
ylog = y(1);
angle = -25;

for i=2:length(x)
    xlog=[xlog,x(i)];
    ylog=[ylog,y(i)];
    delt = t(i)-t(i-1); % Δ t

    % get change for displace
    [change_pos,change_P] = get_change(angle,i,xlog,ylog,delt);
    % move robot to new position (propagation)
    [r_pos,r_P] = propagation(change_pos,change_P,r_pos,r_P);

    % if first landmark, add to map directly
    if i<3
        % add statas(the first landmark) to map
        [disp_pos,disp_P,map_pos,map_P] ...
        =add2state(change_pos,change_P,r_pos,r_P,disp_pos,disp_P,map_pos,map_P);
    else
    % check if it is new state(data association)
    function [label,corl] = ifnewstate(map_pos,map_P,r_pos,r_P);
    % if new states, add to map, change angle and continue
    if label == 0
        [disp_pos,disp_P,map_pos,map_P] ...
        =add2state(change_pos,change_P,r_pos,r_P,disp_pos,disp_P,map_pos,map_P);
            corner_type = cornerlog(i-1);
            [angle,¬,idx,cornerlog] = ...
                angle_change(angle,corner_type,i-1,cornerlog);
            angle = mod(angle, 360);
    else % loop closure
        % intersection in current position with associated landmark
        [r_pos,r_P] = intersection(map_pos(:,corl),r_pos,map_P(:,corl),r_P)

        back_pos = r_pose;
        back_P = r_P; % first pose and error after correction

    % first propagation and fusion
    % propagation
    [change_pos,change_P] = back_change(angle-130,i,xlog,ylog,t(i)-t(i-1));
```

```
    % b_pos and back_P is the robot pose and error through back propagation
    [b_pos,b_P] = propagation(change_pos,change_P,r_pos,r_P);
    % fusion
    [b_pos,b_P] = intersection(b_pos,map_pos(:,corl-1),b_P,map_P(:,corl-1));
    back_pos = [back_pos,b_pose];
    back_P = [back_P,b_P];
    angle = angle_back(angle,i-1,cornerlog);
        break;
        end
    end
end

j=i-1;
while inte==1   % back propagation and fusion till there is no intersection ...
    of bounding box, or to initial robot pose
    [change_pos,change_P] = back_change(angle,i,xlog,ylog,t(j)-t(j-1));
    [b_pos,b_P] = propagation(change_pos,change_P,b_pos,b_P);
    [b_pos,b_P] = intersection(b_pos,map_pos(:,j-1),b_P,map_P(:,j-1));
    back_pos = [back_pos,b_pose];
    back_P = [back_P,b_P];
    angle = angle_back(angle,j-1,cornerlog);

    if j==2
    break;
    end
    j = j - 1;
end
% plot
plotfigure(map_pos,map_P);
```

get change.m

```
% This function is for getting the displacement of robot movement
% and error according to this movement from a corner to another corenr.
%
% Input:
%   angle    -  Angle betewen robot heading and initial wall (positive in ...
    clockwise)
%   i        -  Looping parameter
%   xlog     -  List of landmark position x
%   ylog     -  List of landmark position y
```

```
%   delt     -  Motion time from corner to corner

% Output:
%   change_pos  -    Updated robot position change (displacement)
%   change_P    -    Updated ovement error (corner to corner)


function   [change_pos,change_P] = get_change(angle,i,xlog,ylog,delt)
    varp = 0.03;
% calculate the measurment data according to x and y aixes :
rtheta = deg2rad(abs(angle));  % rtheta is the rubot running heading angle ...
    in world frame,rad

if  abs(sin(rtheta))*abs(xlog(end)-xlog(end-1)) ...
≥abs(cos(rtheta))*abs(ylog(end)-ylog(end-1))
    if  rtheta < pi % x increasing
        % measurment accumulated
        disx =  + abs(xlog(end)-xlog(end-1));
        disy = 0;
    else  % x decreasing
        % measurment accumulated
        disx = - abs(xlog(end)-xlog(end-1));
        disy = 0;
    end
    q = varp*delt+0.5;
    change_P = [q,0;0 5000000];
else
    if  rtheta < pi/2 || rtheta > 3*pi/2 % y increasing
        % measurment accumulated
        disx = 0;
        disy = + abs(ylog(end)-ylog(end-1));
    else
        disx = 0;
        disy = - abs(ylog(end)-ylog(end-1));
    end
    q = varp*delt+0.5;
    change_P = [5000000 0;0 q];
end
change_pos = [disx;disy];
end
```

propagation.m

```
% this function is to propagate in order to get current robot pose and error.
% Input:
%    change_pos  -    Robot position change (displacement)
%    change_P    -    movement error (corner to corner)
%    r_pos    -    Current robot position
%    r_P      -    Current robot error

% Output:
%    r_pos    -    Updated current robot position
%    r_P      -    Updated current robot error

function  [r_pos,r_P] = propagation(change_pos,change_P,r_pos,r_P)
    r_pos = r_pos + change_pos;
    % ellips to propagate
    P0 = [r_P/2,zeros(2);zeros(2),change_P/2];
    % compute F
    A = [eye(2),eye(2)];
    Ar = A'/(A*A');
    NA = null(A);
    C = -eye(2);
    B0 = NA'*P0*NA;
    B1 = P0*NA*(pinv(B0))*NA'*P0;
    r_P = C'*( (Ar)' )* ( P0 - B1 )*Ar*C;
end
```

intersection.m

```

% This function is to find the intersection of two ellipsoids.

% Input
%    x1  -   Center position of one bounding box
%    x2  -   Center position of another bounding box
%    E1  -   Error of one bounding box
%    E2  -   Error of another bounding box
% Output
%    E   -   Error after intersection
%    x0  -   Error after intersection
```

```

function [E,x0] = intersection(x1,x2,E1,E2)
    w = sqrt( (x1(1))^2 + x2(1)^2 ) + sqrt( (x1(1))^2 + x2(1)^2 );
    if w < 20
        lambda = 0.5;
        X = lambda*E1+(1-lambda)*E2;
        k = 1-lambda*(1-lambda)*(x2-x1)'*E2* (X^(-1)) *E1*(x2-x1);
        E = 1/k*X;
        x0 = (X^(-1)) * (lambda*E1*x1+(1-lambda)*E2*x2);
     end
end
```

plotellipse.m

```
% This function is to plot the ellispsoid (error)
% Input
% x - center of ellipsoid
% E - error(ellipsoid)
function plotellipse(x,E)
    a = 0:0.01:2*pi;
    c = cos(a);
    s = sin(a);
    d = x+E\[s;c];
    plot(d(1,:),d(2,:));
end
```

angle change.m

```
%% This function is to update the angle according to the robot movement
% Input:
%    angle    -  angle beteween robot heading and initial wall (positive in ...
     clockwise)
%    corner_type -  corner type signal
%    idx      -  pointer of landmark (-th wall)
%    cornerlog    -  List of all the corner type from the initial robot position
% Output:
%    angle    -  angle beteween robot heading and initial wall (positive in ...
     clockwise)
%    corner_type -  updated corner type signal
%    idx      -  updated pointer of landmark (-th wall)
```

```
%   cornerlog   — List of all the corner type from the initial robot position

function [angle,corner_type,idx,cornerlog] ...
=angle_change(angle,corner_type,idx,cornerlog)
    if corner_type ==1  % convex corner
        angle = angle + 90;
    else
        if corner_type == 2  % concave corner
            angle = angle — 90;
        end
    end
    cornerlog=[cornerlog,corner_type];
    corner_type = 0; % reset corner_type

    % set angle to the range ( 0—360 )
    angle = mod(angle, 360);

    % increase the index
    idx = idx+1;
end
```

Bounding box approach

bounding main.m

```
%% main run simulation with record datas
% This function is the main function for Bounding box approach
% back_pos and back_P are the states and the covariance after loop—closing
% map_pos and mao_P are the states and the covariance before loop—closing

% initialization
label=0;
inte=0;
% robot
r_pos = [0;0] ;
r_P = [0.1;0.1]; %initial error;

change_pos = [];
change_P = [];
```

```
disp_pos = [0;0]; % displacment
disp_P = [0;0];

map_pos = [0;0];  % initial map with the current robot position
map_P = [0.1;0.1]; %initial error;


% load('**.mat');
% load('cornerdatabigmap.mat');

xlog = x(1);
ylog = y(1);
angle = -25; % initial angle for exploration

for i=2:length(x)
    xlog=[xlog,x(i)];% sensor position x
    ylog=[ylog,y(i)];% sensor position y
    delt = t(i)-t(i-1); % ∆ t

    % get change for displace
    [change_pos,change_P] = get_change(angle,i,xlog,ylog,delt);
    % move robot to new position (propagation)
    [r_pos,r_P] = propagation(change_pos,change_P,r_pos,r_P);

    % if first landmark, add to map directly
    if i<3
       [disp_pos,disp_P,map_pos,map_P]= ...
       add2state(change_pos,change_P,r_pos,r_P,disp_pos,disp_P,map_pos,map_P);
       corner_type = cornerlog(i-1);
       [angle,¬,idx,cornerlog] = ...
       angle_change(angle,corner_type,i-1,cornerlog);
       angle = mod(angle, 360);
     else
    % check if it is new state(data association)
    [label,corl] = ifnewstate(map_pos,map_P,r_pos,r_P);
%   % if new states, add to map, change angle and continue
        if label == 0
            [disp_pos,disp_P,map_pos,map_P] = ...
                add2state(change_pos,change_P,r_pos,r_P, ...
            disp_pos,disp_P,map_pos,map_P);
            corner_type = cornerlog(i-1);
```

```
                [angle,¬,idx,cornerlog] = ...
                    angle_change(angle,corner_type,i−1,cornerlog);
                angle = mod(angle, 360);
            else % loop closure
                % intersection in current position with associated landmark
                [b_pos,b_P,¬] = ...
                    intersection(map_pos(:,corl),r_pos,map_P(:,corl),r_P);
                back_pos = b_pos;
                back_P = b_P; % first pose and error after correction
                % first propagation and fusion
                % propagation
                [change_pos,change_P] = ...
                    back_change(angle−130,i,xlog,ylog,t(i)−t(i−1));
                % b_pos and back_P is the robot pose and error through back ...
                    propagation
                [b_pos,b_P] = propagation(change_pos,change_P,r_pos,r_P);
                % fusion
                [b_pos,b_P,inte] = ...
                    intersection(b_pos,map_pos(:,corl),b_P,map_P(:,corl));
                back_pos = [back_pos,b_pos];
                back_P = [back_P,b_P];
                angle = angle_back(angle,i−1,cornerlog);
            end
        end
end

j=i−1;
while inte==1   % back propagation and fusion till there is no intersection ...
    of bounding box, or to initial robot poses
    [change_pos,change_P] = back_change(angle,i,xlog,ylog,t(j)−t(j−1));
    [b_pos,b_P] = propagation(change_pos,change_P,b_pos,b_P);
    [b_pos,b_P,inte] = intersection(b_pos,map_pos(:,j−1),b_P,map_P(:,j−1));
    back_pos = [back_pos,b_pose];
    back_P = [back_P,b_P];
    angle = angle_back(angle,j−1,cornerlog);

    if j==2
    break;
    end
    j = j − 1;
end
```

```

% plot
figure,plotfigure(map_pos,map_P);
figure,plotfigure(back_pos,back_P); % after loop closure
```

get change.m

```
% This function is for getting the displacement of robot movement
% and error according to this movement from a corner to another corenr.
%
% Input:
%    angle    -  Angle beteween robot heading and initial wall (positive in ...
    clockwise)
%    i        -  Looping parameter
%    xlog     -  List of landmark position x
%    ylog     -  List of landmark position y
%    delt     -  Motion time from corner to corner

% Output:
%    change_pos  -    Updated robot position change (displacement)
%    change_P    -    Updated ovement error (corner to corner)

function     [change_pos,change_P] = get_change(angle,i,xlog,ylog,delt)
    varp = 25;
    % calculate the measurment data according to x and y aixes :
    rtheta = deg2rad(abs(angle));  % rtheta is the rubot running heading ...
        angle in world frame,rad

    if  abs(sin(rtheta))*abs(xlog(end)-xlog(end-1))≥ ...
    abs(cos(rtheta))*abs(ylog(end)-ylog(end-1))
        if  rtheta < pi % x increasing
            % measurment accumulated
            disx =  + abs(xlog(end)-xlog(end-1));
            disy = 0;
        else  % x decreasing
            % measurment accumulated
            disx = - abs(xlog(end)-xlog(end-1));
            disy = 0;
        end
        q = varp*delt+7; % error
        changeq= [q;0];
```

```
    else
        if  rtheta < pi/2 || rtheta > 3*pi/2 % y increasing
            % measurment accumulated
            disx = 0;
            disy = + abs(ylog(end)−ylog(end−1));
        else
        disx = 0;
        disy = − abs(ylog(end)−ylog(end−1));
        end
        q = varp*delt+7;  % error
        changeq = [0;q];
    end
        change_P = changeq;
        change_pos = [disx;disy];
end
```

propagation.m

```
%  this function is to propagate in order to get current robot pose and error.
% Input:
%   change_pos  −    Robot position change (displacement)
%   change_P    −    movement error (corner to corner)
%   r_pos   −    Current robot position
%   r_P     −    Current robot error

% Output:
%   r_pos   −    Updated current robot position
%   r_P     −    Updated current robot error

% adapt the robot pose and the error
function    [r_pos,r_P] = propagation(change_pos,change_P,r_pos,r_P)
    r_pos = r_pos + change_pos;
    r_P = change_P + r_P;
end
```

ifnewstate.m

```
% This function is to check if the current position is associated to one ...
    mapped landmark

```

```
% Input:
%   map      — Map (states, include the current robot position)
%   P        — Error (include the current robot covariance)
%       r_pos   —    Current robot position
%       r_P —    Current robot error
% Output:
%   label    — signal for data association   label = 0 no associated ...
    landmark(new state);
%                         label = 1 landmark associated
%   corl     — pointer of associated landmark (—th) in the state

function [label,corl] = ifnewstate(map_pos,map_P,r_pos,r_P)
    % initialization
    label = 0;
    corl = 0;
    W = 0; % similarity score [0—1]
    temW = 0; % temporary value for comparation


        % find maximum W
    for i=1:length(map_pos)—1
        [S3E,x3,inte] = intersection(map_pos(:,i),r_pos,map_P(:,i),r_P);
        if inte == 1
            % area: min (S1,S2), because of propagation——
            % ——> landmark always smaller error than current robot pose
            E = map_P(:,i);
            area = E(1)*E(2);
            S3 = S3E(1)*S3E(2);
            temW = S3/area;
                if temW > W
                W = temW;
                    corl = i;
                end
        end
    end
    if W>0.8
        label = 1;
    end
end
```

add2state.m

```
% This function is update the map, error and displacement list.
% Input:
%   change_pos  —   Robot position change (displacement)
%   change_P    —   Movement error (corner to corner)
%   r_pos       —   Current robot position
%   r_P     —   Current robot error
%   disp_pos    —   Displacement list
%   disp_P      —   Movement error list
%   map_pos     —   Map (landmark list)
%   map_P       —   Error list (landmark error)
%
% Output:
%   disp_pos    —   Updated displacement list
%   disp_P      —   Updated movement error list
%   map_pos     —   Updated Map (landmark list)
%   map_P       —   Updated error list (landmark error)

function [disp_pos,disp_P,map_pos,map_P]= ...
add2state(change_pos,change_P,r_pos,r_P,disp_pos,disp_P,map_pos,map_P)
    disp_pos = [disp_pos,change_pos];
    disp_P = [disp_P,change_P];

    map_pos = [map_pos,r_pos]
    map_P = [map_P,r_P];

end
```

angle change.m

```
%% This function is to update the angle according to the robot movement
% Input:
%   angle    —  angle beteween robot heading and initial wall (positive in ...
    clockwise)
%   corner_type —  corner type signal
%   idx      —  pointer of landmark (—th wall)
%   cornerlog   —  List of all the corner type from the initial robot position
% Output:
%   angle    —  angle beteween robot heading and initial wall (positive in ...
    clockwise)
%   corner_type —  updated corner type signal
```

```
%   idx      -  updated pointer of landmark (-th wall)
%   cornerlog   -  List of all the corner type from the initial robot position

function [angle,corner_type,idx,cornerlog]= ...
angle_change(angle,corner_type,idx,cornerlog)
    if corner_type ==1  % convex corner
        angle = angle + 90;
    else
        if corner_type == 2  % concave corner
            angle = angle - 90;
        end
    end
    cornerlog=[cornerlog,corner_type];
    corner_type = 0; % reset corner_type

    % set angle to the range ( 0-360 )
    angle = mod(angle, 360);

    % increase the index
    idx = idx+1;
end
```

intersection.m

```
% This function is to find the intersection of two bounding box.
% Before doing intersection, we need to make sure those two box are associated.

% Input
%   x1  -   Center position of one bounding box
%   x2  -   Center position of another bounding box
%   E1  -   Error of one bounding box
%   E2  -   Error of another bounding box
% Output
%   E   -   Error after intersection
%   x0  -   Error after intersection
%   inte    -   Symbol of intersection
%           inte = 0 --> two boxes no intersection
%           inte = 1 --> two boxes have intersection

function [E,x0,inte] = intersection(x1,x2,E1,E2)
        inte = 0; % there is no intersection
```

```

    xmin1 = x1(1)-1/2*E1(1);
    ymin1 = x1(2)-1/2*E1(2);  % x1 left down corner x and y
    xmin2 = x2(1)-1/2*E2(1);
    ymin2 = x2(2)-1/2*E2(2);  % x2 left down corner x and y

    xmin0 = xmin2;
    xmax0 = xmin2 + E2(1);
    ymin0 = ymin2;
    ymax0 = ymin2 + E2(2);  % set intersection first as box 1

    % makesure there is intersection
    L1 = abs( x1(1) - x2(1));
    L2 = abs( x1(2) - x2(2));
    D1 = 1/2*( abs(E1(1)) + abs(E2(1)) );
    D2 = 1/2*( abs(E1(2)) + abs(E2(2)) );

    if L1<D1 && L2<D2 % x direction intersection
    inte = 1;
        if (xmin2 > xmin1) && (xmin2 < xmin1+E1(1))
            222
            xmin0 = xmin2;
        end   % left down corner x

        if (xmin2+E2(1) > xmin1) && (xmin2+E2(1) < xmin1+E1(1))
            xmax0 = xmin2+E2(1);
        end   % right down corner x

    inte = 1;
        if (ymin2 > ymin1) && (ymin2 < ymin1+E1(2))
            xmin0 = xmin2;
        end   % left down corner y

        if (ymin2+E2(2) > ymin1) && (ymin2+E2(2) < xmin1+E1(2))
            ymax0 = ymin2+E2(2);
        end  % right down corner y
    end
    E = [(xmax0-xmin0);(ymax0-ymin0)];
    x0 = [xmin0+1/2*E(1);ymin0+1/2*E(2)];
end
```

angle back.m

```
% This function is for change robot heading angle through back propagation
% Input:
%    angle    -  angle beteween robot heading and initial wall (positive in ...
    clockwise)
%    idx      -  pointer of landmark (-th wall)
%    cornerlog   -  List of all the corner type from the initial robot position
% Output:
%    angle    -  angle beteween robot heading and initial wall (positive in ...
    clockwise)


function angle = angle_back(angle,idx,cornerlog)
    if cornerlog(idx) == 1
        angle = angle - 90;
    else
       if cornerlog(idx) == 2
            angle = angle + 90;
       end
    end

    % set angle to the range ( 0-360 )
    angle = mod(angle, 360);

end
```

back change.m

```
% This function is for back propagation to get the change of robot pose and ...
    error.
% Input:
%    angle    -  Angle beteween robot heading and initial wall (positive in ...
    clockwise)
%    i        -  Looping parameter
%    xlog     -  List of landmark position x
%    ylog     -  List of landmark position y
%    delt     -  Motion time from corner to corner

% Output:
```

```
%   change_pos   —    Updated robot position change (displacement)
%   change_P     —    Updated ovement error (corner to corner)

function   [change_pos,change_P] = ...
back_change(angle,i,xlog,ylog,delt)
    varp = 0.15;
    % calculate the measurment data according to x and y aixes :
    rtheta = deg2rad(abs(angle));  % rtheta is the rubot running heading ...
        angle in world frame,rad

    if  abs(sin(rtheta))*abs(xlog(i-1)-xlog(i))≥ ...
    abs(cos(rtheta))*abs(ylog(i-1)-ylog(i))
        if  rtheta < pi % x increasing
            % measurment accumulated
            disx =  + abs(xlog(i-1)-xlog(i));
            disy = 0;
        else  % x decreasing
            % measurment accumulated
            disx = - abs(xlog(i-1)-xlog(i));
            disy = 0;
        end
        q = varp*delt+0.5;
        changeq= [q;0];
    else
        if  rtheta < pi/2 || rtheta > 3*pi/2 % y increasing
            % measurment accumulated
            disx = 0;
            disy = + abs(ylog(i-1)-ylog(i));
        else
            disx = 0;
            disy = - abs(ylog(i-1)-ylog(i));
        end
        q = varp*delt+0.5;
        changeq = [0;q];
    end
        change_P = changeq;
        change_pos = [disx;disy];
end
```

plotfigure.m

```
% This function is to plot the map and error
% Input
%   map_pos    —   center of the box (landmark position)
%   map_P      —   error
function plotfigure(map_pos,map_P)

%   % simple environment without the ball length
%   M00 = [0    0    71.3  71.3   0
%                    0    46.3   46.3   0   0];
    % complex environment without the ball length
     M00 = [0    0    79.5     79.5   207.5    207.5  152.2   152.2    0
                  0   39    39      101     101     60       60      0   0];

    n=length(map_pos);
    plot(M00(1,:),M00(2,:),'r');  % red is the real map
    hold on
    grid on
    plot(map_pos(1,:),map_pos(2,:),'b');  % blue is after the karman
    %% call function 'plotbox' to plot the error
    for i=1:n
         plotbox(map_pos(:,i), map_P(:,i));
    end
end
```

plotbox.m

```
% This function is to plot the bounding box (error)
% Input
%   x     —   center of the box
%   E     —   error
function plotbox(x,E)
    a = x(1)-1/2*E(1);
    b = x(2)-1/2*E(2);
    rectangle('Position',[a b E(1) E(2)]);
end
```

Bibliography

[1] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[2] Ricardo Carelli and Eduardo Oliveira Freire. Corridor navigation and wall-following stable control for sonar-based mobile robots. *Robotics and Autonomous Systems*, 45(3-4):235–247, 2003.

[3] Mauro Di Marco, Andrea Garulli, Antonio Giannitrapani, and Antonio Vicino. A set theoretic approach to dynamic robot localization and mapping. *Autonomous robots*, 16(1):23–47, 2004.

[4] Jon Gallant. How to control a sphero sprk+ with a raspberry pi 3 and node.js. `https://blog.jongallant.com/2016/08/sphero-sprkplus-rpi-nodejs/`. August 29, 2016.

[5] Jay K Hackett and Mubarak Shah. Multi-sensor fusion: a perspective. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1324–1330. IEEE, 1990.

[6] Luc Jaulin. A nonlinear set membership approach for the localization and map building of underwater robots. *IEEE Transactions on Robotics*, 25(1):88–98, 2009.

[7] Srinivas Kandasamy. Slamming with spheros: An impact-based approach to simultaneous localization and mapping. 2015.

[8] Josep M Porta. Cuikslam: A kinematics-based approach to slam. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2425–2431. IEEE, 2005.

ETSEIB

[9] Lluís Ros, Assumpta Sabater, and Federico Thomas. An ellipsoidal calculus based on propagation and fusion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(4):430–442, 2002.

[10] Joris Sijs and Mircea Lazar. State fusion with unknown correlation: Ellipsoidal intersection. *Automatica*, 48(8):1874–1878, 2012.

[11] Joan Sola. Simulataneous localization and mapping with the extended kalman filter. *Avery quick guide with MATLAB code*, 2013.

ETSEIB