EMARO M1
European Master on Advanced Robotics

Master ARIA M1
Automatique, Robotique et Informatique Appliquée
Master in Control Engineering, Robotics and Applied Informatics

Project Report
25th June, 2014

Exploring Kalman Filter Localization with KheperaIII in ROS-Player

Pham Huy Nguyen Nguyen Dong Hai Phuong

Supervisor
Dr. Gaetan Garcia

Table of Contents

Chapter	Title	Page
	Table of Contents	i
	List of Figures	ii
	List of Tables	iii
1	Introduction	1
	1.1 Background and Problem Statement	1
	1.2 Objectives	1
	1.3 Project Report Outline	1
2	Literature Review	2
	2.1 Equipment	2
	2.2 Localization	4
	2.3 ROS	10
	2.4 Player-ROS	11
3	Implementation	12
	3.1 Implementing Player-ROS	12
	3.2 Implementing the Line Extraction	13
	3.3 Implementing the Extended Kalman Filter	17
4	Experiments and Observations	19
	4.1 Experiment configuration	19
	4.2 Experiments	20
	4.3 Results	20
5	Conclusion	27
	References	28
	Appendixes	29

List of Figures

Figure	Title	Page
2.1	The Khepera III	2
2.2	The Korebot II	3
2.3	The Wifi Adapter in the Korebot II	3
2.4	The Hokuyo URG-04LX-UG01 Laser Range Finder	4
2.5	The Building Blocks of Navigation	5
2.6	The Two Stages of Localization	6
2.7	The Schematic for Kalman Filter Localization	7
3.1	Playerros node	13
3.2	Line Extract node	17
3.3	Extended Kalman Filter Localization node	18
4.1	Corner set up for experiment	19
4.2	Running in a circle with radius of 0.3m from initial posture of (0.48, 0.805, $-\pi$) with good estimation of system noise and input noise: left:trajectory, right: Estimated variances	21
4.3	Running in a circle with radius of 0.3m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) with good estimation of system noise and input noise: left:trajectory, right: Estimated variances	22
4.4	Running in a circle with radius of 0.3m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) without laser signal	23
4.5	Running in a circle with radius of 0.3m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) with wrong estimation of system noise: left:trajectory, right: Estimated variances	24
4.6	Running in a circle with radius of 0.3m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) with wrong estimation of input noise: left:trajectory, right: Estimated variances	25
4.7	Trajectory of Khepera on rviz tool with wrong estimation: left:system noise, right: input noise	26
4.8	Trajectory of Khepera on rviz tool with good estimation: left:start at (0.1, 0.4, $-\frac{\pi}{2}$), right: (0.48, 0.805, $-\pi$)	26

List of Tables

Table **Title** **Page**

Chapter 1

Introduction

1.1 Background and Problem Statement

ECN (Ecole Centrale de Nantes) recently acquired two new robots into their family, the KheperaIII and the Baxter. With the opportunity to be the first students to work with the robots, we jumped at the opportunity to do so.

With the knowledge from courses like SOFAR (Software Architecture for Robotics) and MOBRO (Mobile Robotics), we picked a project on implementing a localization method for the KheperaIII.

The KheperaIII robot came with a KoreBotII and a Hokuyo URG-04LX-UG01 Laser Range Finder. The KoreBotII is a embedded PC board that runs Linux to allow you to program the KheperaIII. The Hokuyo Laser Range Finder was to be used as our sensor for localizing the robot.

The KheperaIII robot is widely used around the world, in universities like Georgia Tech, EPFL, and ETH, because of its accurate odometry, its extended capabilities, its small size. However, the KheperaIII is not supported by ROS (Robot Operating System). ROS has many capabilities in the realm of robotics that includes its own Kalman filter and SLAM packages.

1.2 Objectives

To address the challenges laid out in the previous section, the main objectives are to:

1. Study the KheperaIII, a (2,0) differential drive robot.
2. Get the KheperaIII robot to work with ROS.
3. Implement our own Kalman filter algorithm in ROS to localize the robot.

1.3 Project Report Outline

Chapter 2 is divided into 4 main sections. The first section discuss about all the equipment used (The KheperaIII, The Hokuyo Laser Range Finder, and the KorebotII). A section discusses the concept of localization and focuses on the concept of extended kalman filter (EKF). The other two sections gives an overview of the capabilities of ROS and Player-ROS.

Chapter 3 presents the implementation of the Kalman Filter in ROS.

Chapter 4 presents the observations and tests made from trying to localize the robot.

Chapter 5 presents the conclusion and directions for future research.

Chapter 2

Literature Review

2.1 Equipment

2.1.1 KheperaIII

The KheperaIII (presented in *Figure 2.1*) is a minature differential drive robot produced by a Swiss company called K-Team with assistance from EPFL's (Ecole Polytechnique Federale de Lausanne) DISAL (Distribute Intelligent Systems and Algorithms Laboratory). It has a diameter of about 130 mm, the height about 70 mm and the weight about 690 g.

Figure 2.1: The Khepera III

The KheperaIII has two motors, a ring of 9 IR (infrared) sensors on the lower layer, and a ring of 5 US (ultrasonic) sensors on the layer above that. It also has a pair of IR sensors pointing downwards called ground sensors.

The KheperaIII fashions a dsPIC 30F5011 microprocessor, which has a frequency of 60 MHz, RAM of 4 kB, and a flash memory of 66 kB.

In this project, none of the IR or US sensors are used. We use the Hokuyo Laser Ranger Finder instead (mentioned in one of the chapters below).

2.1.2 KorebotII

The KheperaIII comes with a KoreBotII board (presented in *Figure 2.2*), which is stacked on top of the robot. The KoreBotII board gives the robot an embedded Linux Operating System to allow application development.

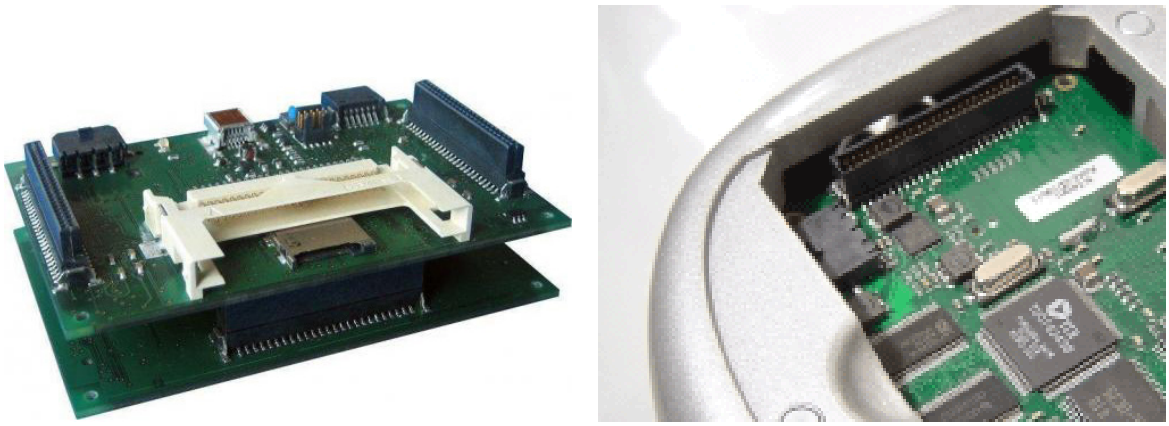

Figure 2.2: The Korebot II

The KoreBotII uses an Intel PXA255 XScale processor running at 400 MHz with a RAM of 60 MB and a 32 MB Flash Memory.

The KorebotII board also has a PCMCIA slot where we inserted the wireless network adapter card as seen in *Figure 2.3*. On top of the KoreBotII board is where we stacked our Hokuyo laser range finder as seen in *Figure 2.4*.

Figure 2.3: The Wifi Adapter in the Korebot II

When the KoreBotII is connected, the dsPIC microprocessor switches from running the communication protocol to the I2C slave mode. So in order to send control commands we have to do it via the

I2C bus. K-Team provides us a library libkorebot (C library) to do so. The access to the ARM Linux command line is now established using the RS 232 connector using the KoreConnect adapter. In our case, we used a standard wireless network to establish connection.

More information on how Player-ROS creates a bridge between the KheperaIII and ROS so we can use ROS as an environment that allows us to compile executable programs, in given below.

2.1.3 Hokuyo URG-04LX-UG01 Laser Range Finder

The Hokuyo URG-04LX-UG01 Laser Range Finder (as seen in *Figure 2.4*) is the laser range finder of choice for the KheperaIII. It was designed so it could be mounted on top of the KoreBotII board as seen in *Figure 2.4*.

It is a small laser scanner that weighs 160 g. It has ranges from 20 mm to 5600 mm with a 1 mm resolution. The scan area is a 240 degree arc with a 0.36 degree angular resolution.

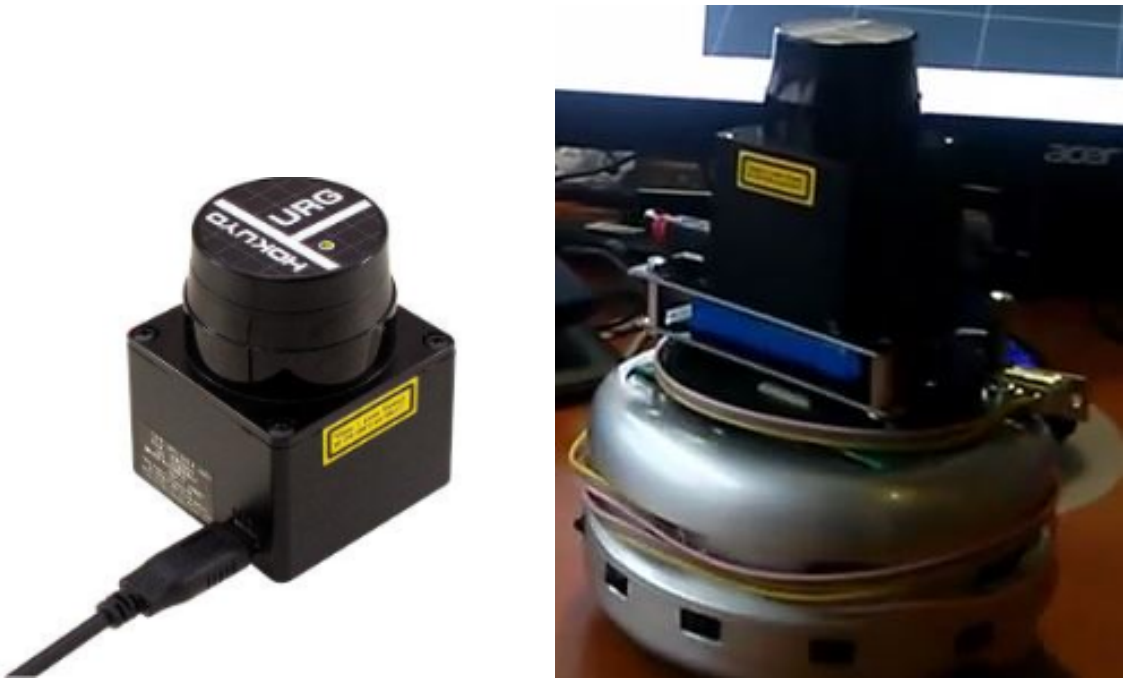

Figure 2.4: The Hokuyo URG-04LX-UG01 Laser Range Finder

2.2 Localization

Navigation is one of the most challenging problems in mobile robotics. Robot navigation has four main building blocks as seen in *Figure 2.5*. *Perception* (working with sensors), *localization* (Robot tracking), *cognition* (Deciding how to achieve its goals), and *motion control* (achieving the desired trajectory using a control algorithm). In our project we deal with localization. Mobile robot localization is the problem of figuring out where the robot is in its environment, i.e. estimate or track its pose (position and orientation) in space.

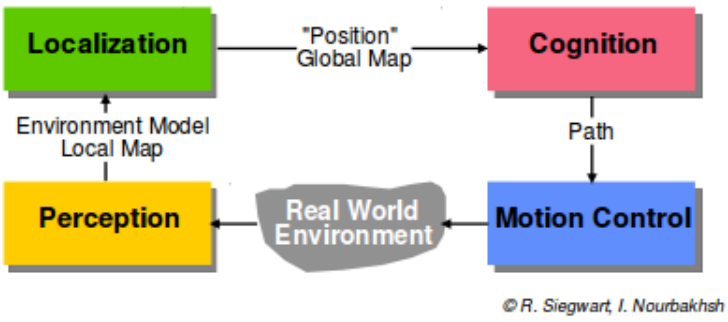

Figure 2.5: The Building Blocks of Navigation

This is a difficult problem because sensors and effectors are key to successful localization (Roland Seigwart & Scaramuzza, 2011). However, using sensors and effectors come with problems like sensor noise, effector noise and sensor aliasing.

Sensors are the the main inputs for the process of perception mentioned above. What is crucial about sensors is that they have to be accurate. Accuracy means they have to be able to discriminate the world state clearly. When sensors cannot capture some environmental features, making these features overlooked, this means there exists sensor noise. Sensor noise comes in many forms, in vision systems we have illumination dependence, picture jitter, signal gain, blooming, and blurring.

Sensor noise reduces the amount of useful information taken by the sensors. To solve this, multiple readings and using multisensor fusion to increase the information content is used.

Effector Noise comes from the robot effectors. In this project, that error comes from the odometry of the robot (e.g. motor encoders). Effectors increase the uncertainty of the robot's future state. This error is because we cannot capture a complete model of the environment. There are a lot of types of odometric error. The wheels might be misaligned, the wheel diameters may be uncertain, the contact point of the wheel might be uncertain, or the floor contact may be unequal causing things like slipping. These unmodeled problems cause a difference between the physical motion of the robot versus the intended motion of the robot since the odometric error is accumulated over time when we integrate the movement of the robot sensed by the encoders in order to find the robot's position.

Sensor Aliasing is the second problems that come with sensors. This problem is only found in robots and not humans. To humans, everything and every place looks different. We have the power to distinguish things that create a unique mapping in most enviornment. However, if we are placed in a place that is completely dark, a place that has no landmarks or cues, a place without visual uniqueness, then our localization system quickly degrades.

Robots face this nonuniqueness in sensor readings, also known as sensor aliasing, all the time. Because there are so many environmental states that create different many-to-one mappings to the robot's perceptual inputs, the robot cannot distinguish these many states. Even with noise-free sensors, the amount of information is insufficient to identify the robot's position with a single-precept reading. That's why the robot must read a series of readings in order to receive sufficient information to correct the robot's pose over time.

So what localization does is that we want to filter out all the types of noise from the data. The Kalman filter is an algorithm that processes all the available measurements, not worrying about their precision,

to esitmate the current pose (position and orientation) of the robot.

To summarize in terms of a general robot localization problem, consider a mobile robot, in our case the KheperaIII, moving in a known environment (we have a map of the environment already). As it starts to move, we keep track of that movement using the robot's odometry. There is uncertainty in the effector (odometry uncertainty) so after a while, the robot will be very uncertain of its position. So what the robot has to do now is localize itself in relation to its current environment using its exteroceptive sensors, in this project a laser range finder is used. The information from the exteroceptive sensors corrects the position of the robot. These two processes are generally separated into two iterative steps as seen in *Figure 2.6*.

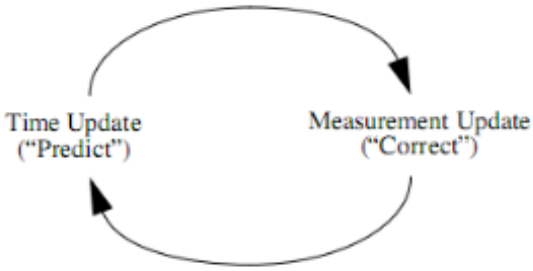

Figure 2.6: The Two Stages of Localization

The *prediction or time update* uses the proprioceptive sensors (wheel encoders) to estimate its motion. The uncertainty increases due to the integration of the odometric error over time. The *perception or measurement or correction update* uses the information from its exteroceptive sensors to correct the position estimated during the prediction phase, shrinking the uncertainty of the robot's pose.

2.2.1 Extended Kalman Filter Localization

In this section, the process of applying the Kalman filter to a differential drive robot is explored.

The Kalman filter began with Prof. Rudolf E. Kalman's paper in 1960. It was made famous when it was used by Stanley Schmidt for non-linear functions creating the Schmidt-Kalman Filter now known as the Extended Kalman Filter (EKF).

The EKF is an extension of the Kalman filter to nonlinear systems. The motion model of a differential drive robot is an example of a non-linear system.

As mentioned above, all localization problems can be divided into two phases, *prediction update* and *correction update*. What separates the Kalman filter from the likes of the Markov filter is that it has a multi-step correction update process.

Figure 2.7 shows a schematic for the Kalman filter localization (Roland Seigwart & Scaramuzza, 2011). First, the *prediction update* occurs, then the *correction update*. The *correction update* is subdivided into multiple steps as follows:

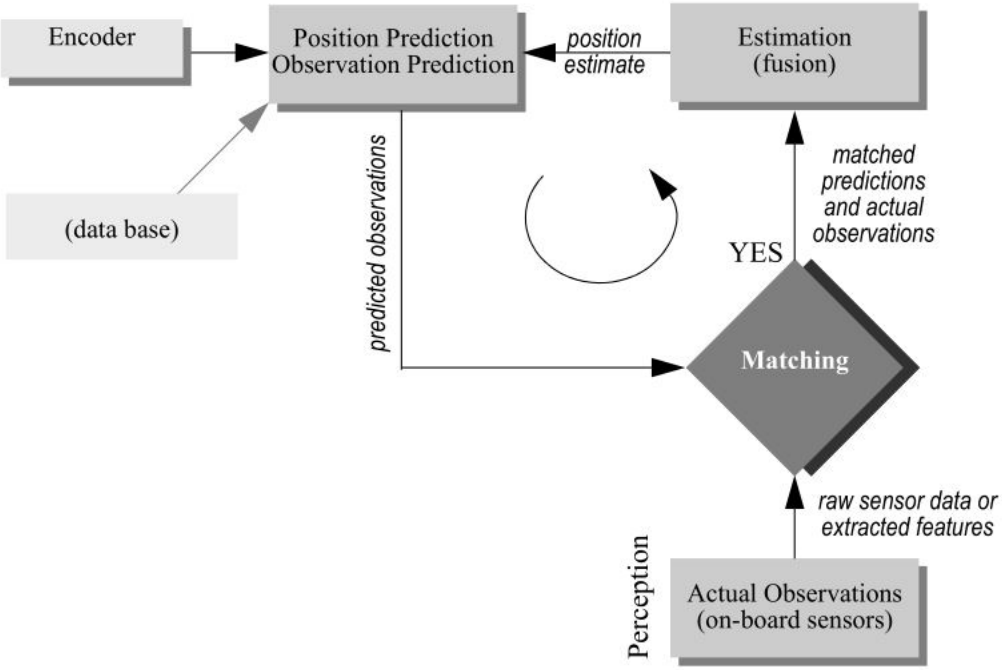

Figure 2.7: The Schematic for Kalman Filter Localization

1. Observation Step - The robot collects data from the sensors and calculates the covariance, which accounts for the quality of the measurements between the measurement points and the line extracted from them.
2. Measurement Prediction - At the same time, based on the robot's prediction position in the map, a measurement prediction is done to find out what the robot expects to see if it were at that particular position.
3. Matching Step - We compare the observation against the predicted observation found in the previous step.
4. Estimation Step - The Kalman gain is calculated and used to estimate the current state of the robot.

Prediction update: Using the motion model of the robot, the robot's position at time *k+1* is predicted by movement due to the control input U_k and its old position at *k*:

$$\widehat{X}_{k+1} = f(X_k, U_k) = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta D_k cos\theta_k \\ y_k + \Delta D_k sin\theta_k \\ \theta_k + \Delta\theta_k \end{bmatrix} \qquad \text{(Equation 2.1)}$$

Where ΔD_k, $\Delta\theta_k$, X_k, and U_k are equal to:

$$\begin{cases} \Delta D_k = v \cdot T_s \\ \Delta\theta_k = \omega \cdot T_s \\ X_k = [x_k, y_k, \theta_k]^T \\ U_k = [v, \omega]^T \end{cases} \qquad \text{(Equation 2.2)}$$

Both v and ω are taken from the robot's odometry.

We then calculate the Jacobian matrices of the robot's position (This step is only found in the EKF, where we calculate the linear approximation of the motion model):

$$A = \frac{\partial f}{\partial X}(X, U) = \begin{bmatrix} 1 & 0 & -\Delta D_k sin(\theta_k) \\ 0 & 1 & \Delta D_k cos(\theta_k) \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(Equation 2.3)}$$

And,

$$B = \frac{\partial f}{\partial U}(X, U) = \begin{bmatrix} T_s cos(\theta_k) & 0 \\ T_s sin(\theta_k) & 0 \\ 0 & T_s \end{bmatrix} \qquad \text{(Equation 2.4)}$$

Where, T_s is the period of the system.

If we take input U_k as a random signal, meaning that the input has a certain amount of noise, then we have U_k^*. U_k^* is basically U_k disturbed by an additive noise β_k. In the same sense, the differences between the real system and the models are represented by the additional additive noises. We have α_k for the evolution equation and γ_k for the observation equation. With this, we modify the equations to:

$$\begin{cases} X_{k+1} = f(X_k, U_k) + \alpha_k \\ U_k^* = U_k + \beta_k \\ Y_{k+1} = g_B(X_k) + \gamma_k \end{cases} \qquad \text{(Equation 2.5)}$$

Knowing the plant and the error model we can compute P_{k+1}, the covariance of the next robot state:

$$\widehat{P}_{k+1} = A_k \cdot P_k \cdot A_k^T + B_k \cdot Q_\beta \cdot B_k^T + Q_\alpha \qquad \text{(Equation 2.6)}$$

Note that Q_α is the covariance of the noise associated to the motion model and the covariance matrix of the additive noise to the input U_k is Q_β.

Measurement update: This stage is divided into four steps mentioned below.

1. Observation This part we observe, Y_j, also known as an observation. The observation varies for the type of localization problem we deal with. For line-based localization, each observation is a line feature extracted from the laser range finder data. So, Y_j = [ϕ_k^i, D_k^i]. Where ϕ is the angle

and D_k is the distance to the line feature from the robot frame. If we deal with a beacon-based or landmark-based localization the ϕ is the angle and D_k is the distance to the beacon from the robot frame.

There are three types of localization strategies based upon what we take Y_j as:

- Using only distance, D_k. This strategy is used in our algorithm.
- Using only angle, ϕ.
- Using both.

Based on this project's localization problem, landmark-based localization, and this project's strategy, using only distance, we have:

$$Y_j = g_B(X_k) = D_k \qquad \text{(Equation 2.7)}$$

Here, D_k is the distance from the robot to the corners measured by the laser range finder.

This leads to covariance matrix Q_γ, which is the uncertainty of all the measurement points contributing to each line (in line-based localization) or to each beacon point (in landmark-based localization).

2. Measurement Prediction In this step we calculate the predicted observation, $\widehat{Y_j}$ based on the predicted robot position $\widehat{X}_k$. For this project (landmark-based localization using only distances):

$$\widehat{Y_j} = g_B(\widehat{X}_k) = \sqrt{(x_c - \widehat{x})^2 + (y_c - \widehat{y})^2} \qquad \text{(Equation 2.8)}$$

And we also find its Jacobian.

$$C = \frac{\partial g_B}{\partial X}(X) = \begin{bmatrix} \frac{\partial D_j}{\partial \widehat{x}} & \frac{\partial D_j}{\partial \widehat{y}} & \frac{\partial D_j}{\partial \widehat{\theta}} \end{bmatrix} = \begin{bmatrix} \frac{\widehat{x}_{k+1} - x_c}{D_k} & \frac{\widehat{y}_{k+1} - y_c}{D_k} & 0 \end{bmatrix} \qquad \text{(Equation 2.9)}$$

3. Matching Matching is trying to find the correspondence between the predicted and the observed features, Y_j and $\widehat{Y_j}$. We do this by calculating the Mahalanobis distance and comparing it to a threshold.

The term **innovation =** $[Y_j - \widehat{Y_j}]$.

$$\Sigma^{ij}_{IN_t} = C^j \cdot \widehat{P}_t \cdot C^{j^T} + Q_\gamma \qquad \text{(Equation 2.10)}$$

To find the best matches while eliminating all the other unmatched features:

$$[Y_j - \widehat{Y_j}]^T \cdot (C^j \cdot \widehat{P}_t \cdot C^{j^T} + Q_\gamma)^{-1} \cdot [Y_j - \widehat{Y_j}] \leq threshold \qquad \text{(Equation 2.11)}$$

Only features where it is less than the threshold are kept.

4. Estimation

Now we calculate the Kalman Gain K_t for only the features that matched in the previous step:

$$K_t = \widehat{P}_t \cdot C_t^T \cdot (C^j \cdot \widehat{P}_t \cdot C^{j^T} + Q_\gamma)^{-1} \qquad \text{(Equation 2.12)}$$

Using the Kalman Gain we update state as follows:

$$x_t = \widehat{x}_t + K_t[Y_j - \widehat{Y}_j] \qquad P_t = \widehat{P}_t - K_t \cdot C \cdot \widehat{P}_t \qquad \text{(Equation 2.13)}$$

2.3 ROS

ROS is a software framework for developing robot applications. Originally developed at Stanford's Artificial Intelligence Laboratory in 2007, the open source, meta-operating system ROS has now been used by so many all around the world. ROS has two major sides, an operating system side and a side that leans more towards implementation. The implementation side includes hardware abstraction, low-level device control, package management, etc. There might be a slow learning curve when it comes to learning ROS but it has its merits. ROS includes a lot of packages that can be used for all sorts of robots, allowing great software reuseability. ROS is very good when it comes to rapid testing, either using rviz as a visualization tool or being able to store and replay sensor data using a tool called rosbag. ROS has many many applicable capabilities for example:

1. Perception
2. Recognition
3. Mobile Robotics
4. Planning and Grasping

This project make good use of a lot of ROS's capabilities. All the code in this project is written using the roscpp package, which allows us to program our nodes and interact with topics, services, and parameters using C++. Other interesting packages we used include tf and rosbag.

rosbag is a tool is ROS that allows us to record messages published by many topics to a file. This file is then used to replay those messages. A good use of rosbag for this project is to record the sequence of the robot moving, the data from laser range finder as it moves, and then replaying the data to use with the loclaization algorithm that is currently under development.

tf is a package that lets us keep track of the coordinate frames we set over time. Basically, it allows us to set multiple coordinate frames to describe the different positions of the various parts of the robot and the different positions of the obstacles it would like to interact with. Then *tf* allows to find the relationship between the different frames set by knowing the transformation that converts coordinates of one frame to another. In this project, we set a world frame from where the positions of the corners are found. We also set a odometer frame, which is the frame of the initial position of the robot. The last frame set is the robot frame that moves and rotates according to the robot.

sensor_msgs, *geometry_msgs*, and *nav_msgs* are message types. *sensor_msgs* defines the message for commonly used sensors, in our case the laser range finder. *geometry_msgs* allows us to send and receive messages about our robot's pose. *nav_msgs* allows messages to interact with the navigation stack related to odometry.

2.4 Player-ROS

The Player Project is a free software for robot research (Project, 2014). Player provides a robot server that uses a client/server model to allow users to write control programs in any langauge on a computer with a network connection to the robot.

Player supports a wide variety of robots and accessories that include the Khepera and the Hokuyo Laser Range Finder.

Player provides the user a simple interface to the robot's sensors and actuators over an IP network. The client program on the other hand, talks to the Player over a TCP socket, reading data from sensors and writing commands to actuators.

What Player does is that it has a Player server that provides interfaces to different robotic devices and sensors. Player communicates with these devices using device drivers. For the Khepera, there is a *khepera driver* to interface with the Khepera robot. Player then provides two interfaces, *position2d* (this interface is used to control the Khepera robot's base) and *ir* or *ranger* (gives access to the IR sensors or the laser range finder).

Since there is no ROS support for the KheperaIII, Player is used to bridge the robot to ROS. The Hokuyo Laser Range Finder uses Player's LaserProxy. What happens is that the *khepera driver* is converted into a ROS node allowing us to control the KheperaIII using ROS. More on this is mentioned in Chapter 3. And the manual on making ROS work with the Khepera III via Player is seen in Appendix B.

Chapter 3

Implementation

3.1 Implementing Player-ROS

The main task of ”playerros” node is creating an ”adapter” between the Player client and ROS network, which are:

- Collecting sensor data in Player client (collected from Khepera by Player server in Khepera), packaging and publishing to ROS network in form of ROS messages, i.e ”/odom” topic containing odometry information of Khepera robot and ”/base_scan” containing laser measurements from Hokuyo Laser Range Finder.
- Subscribing to ROS message ”/cmd_vel” topic, containing velocity commands, from ROS network and sending velocity information to Player server running in Khepera through wifi signal.

The information from Khepera to ROS is bypassed through this Subcriber-Publisher node rather than processed. All subscribe and publish action are asynchronous.
In the ROS part, this node uses:

- ”nav_msgs::Odometry” message type to publish the odometry information
- ”sensor_msgs::LaserScan” message type to publish the laser scan information
- ”geometry_msgs::Twist” message type to subscribe to velocity commands

In the Player part, the node uses:

- ”PlayerCc::Position2dProxy” proxy type to get the velocity and position information from the robot and set the desired velocity to robot.
- ”PlayerCc::LaserProxy” proxy type to get the laser range finder data: angles and distances of all visible measurements from Laser Range Finder to objects as well as Laser Range Finder parameters.

In terms of Transformation tree, this node creates a basis tree to represent the relation between physical parts used in the whole system, including:

- ”odom” frame: is a fixed frame at the initial position of the robot. The relation between this frame and the global frame ”world” will be defined by the initial pose of the robot to the ”world” frame. This part will be explained more in the launch file part.
- ”base_link” frame: is a frame attached to the base of Khepera, which will move with respect to the ”odom” frame and ”world” frame. The relation of this frame and ”odom” frame is defined by the posture (x, y, θ) of the Khepera with respect to the initial position.

- ”base_laser” frame: is a frame attached to the Hokuyo Laser Range Finder of Khepera. The relation of this frame and ”base_link” frame is defined by position of this device in robot frame: i.e, no translation along x and y axis, no rotation around z axis.

Summary of this node can be described in *Figure 3.1*:

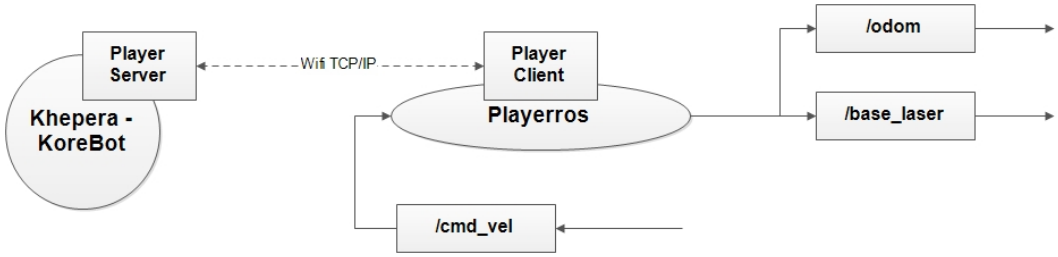

Figure 3.1: Playerros node

One notice is that the difference between the firmware used in our Khepera and the one used to build Player driver for Khepera (by K-Team Company) creates the discrepancies between the reported velocity from the Player driver and the real measure one. Thus causes the very poor odometry and requires some calibration!

3.2 Implementing the Line Extraction

The main task of ”line_extract” node is processing the laser data from Laser Range Finder generate to useful information for Kalman filter:

- Subscribing to the ROS message ”/base_scan” containing laser measurements from Hokuyo Laser Range Finder.
- Fitting points of objects obtained by Laser Range Finder to linear features defining the real environment around Khepera (because Khepera only works in indoor environment which mostly combines of lines)
- Estimating the corners in the environment and extracting the real measurements corresponding to those corners.
- Publishing the ROS message ”/extract_corners” (type ”geometry_msgs::PolygonStamped”) containing distance and angle of detected corners in the environment.
- Publishing the ROS message ”/laser_marker” (including line segments and extracted corners) for visualization in rviz tool.

In order to cope with the above tasks, the laser data has to be stored and divided into many segments corresponding to real lines in the environment, then be fitted in linear curves (in form of y = ax+b, with (x,y) is coordinate of each point in the line) by using least squares method. These line segments are also defined by 2 laser points, one at the beginning and one at the end of the segment. The detected

corners in the laser data set are the intersections between two successive line segments. A valid corner i-th is an intersection which is not too far from the last point of k-th line segment and first point of (k+1)-th line segment (these two segments really cut each other). Part of this algorithm is from ”Split-and-Merge Algorithm” studied in (Nguyen, Martinelli, Tomatis, & Siegwart, 2005),(Teslic, Klancar, & Skrjanc, 2008). The whole laser data processing algorithm can be described in the following pseudo-code:

Algorithm 1 Line Extraction

```
procedure STORE LASER DATA
    ranges[i] ← laser_msg.ranges[i] (0 ≤ laser_message.ranges[i] ≤ MaxRange)
    angle[i] ← laser_msg.angle_increment*i + laser_msg.angle_min
    LaserPoints[i] ← x[i],y[i]
end procedure
procedure SEGMENTATION
    if distance(point(k),point(k+1) ≤ Threshold then
        Segment containing point(k) ← point(k+1)
    else
        New Segment ← point(k+1)
    end if
end procedure
procedure LINE FITTING BY LEAST SQUARE METHOD
end procedure
procedure RE-SEGMENTATION
    hasNewSegment ← true
    while hasNewSegment = true do
        MaxDist ← Max of distance(point[k],Segment[i]), point[k] ∈ Segment[i]
        if MaxDist ≤ Threshold then
            do nothing
        else
            Segment[i] ← New Segment[i] with lastPoint ← point[k]
            Insert New Segment[i+1] with firstPoint = point[k+1] of old Segment[i]
```

Algorithm 2 Line Extraction Continued...

```
            hasNewSegment ← true
            Line Fitting: Segment[i] and Segment[i+1]
        end if
    end while
end procedure
procedure FILTER OUT SEGMENTS WITH LENGTH EQUAL 0
end procedure
procedure FILTER OUT SEGMENTS WITH LENGTH EQUAL 0
end procedure
procedure FIND CORNERS
    corner ← intersection(Segment[i],Segment[i+1])
    if corner = VALID then
        Corners[k] ← corner
    end if
end procedure
procedure FIND READ MEASUREMENT OF CORNER
    if Distance(Corners[k],LaserPoints[i])≤Threshold then
        cornersDist[k] ← ranges[i]
        cornersAngle[k] ← angle[i]
    end if
end procedure
```

The coordinate of intersection point between two successive segment k and (k+1) can be calculated as follows:

$$\begin{cases} x &= \frac{b_{k+1}-b_k}{a_k-a_{k+1}} \\ y &= \frac{a_k.b_{k+1}-a_{k+1}.b_k}{a_k-a_{k+1}} \end{cases} \qquad \text{(Equation 3.1)}$$

with (a_k, b_k) and (a_{k+1}, b_{k+1}) parameters of line segment k and (k+1) respectively, obtained by line fitting procedure. Summary of this node can be described in the *Figure 3.2*:

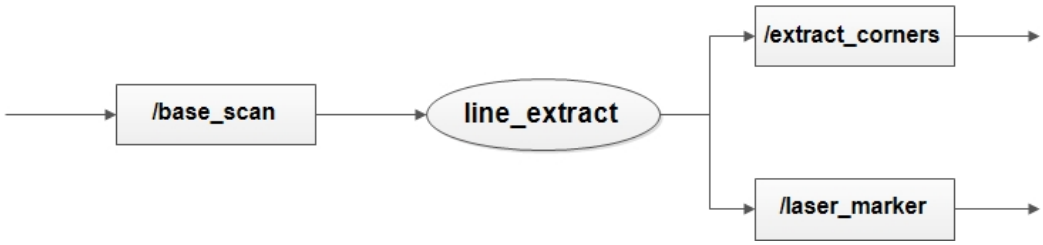

Figure 3.2: Line Extract node

3.3 Implementing the Extended Kalman Filter

The main task of "ekf_kalman" node is processing the odometry data from Khepera and laser data from Hokuyo Laser Range Finder and providing an estimation of robot posture by using the Extended Kalman algorithm discussed in Literature part. This node will:

- Subscribe to ROS message "/odom" from the Playerros node to get the velocity (linear and angular) information of robot.
- Subscribe to ROS message "/extract_corners" from the Line_extract node to get real measurements (distance and angle) of the extracted corners in the environment.
- Apply Kalman filter algorithm to data
- Publish ROS message "/odom_kalman_predict" of filtered posture of robot and "/odom_kalman_estimate" to announce the appearance of good measurement to correct robot posture . The publishing of two separate Odometry messages as above is only for easiness of visualization of correction stage of Kalman filter.

In order to handle the above task, the node program is structured as follows:

- Using a class named "mobroEKF" to deal with all input, output and calculation procedures for Kalman filter of mobile robot in differential motion mode.
- Reading the configuration XML file to get setting parameter for Khepera and Kalman filter: initial posture, noise covariance.
- Reading the map XML file to get the position of all corners in the map.
- Updating the velocity information and duration (between two successive update times) to the filter of Khepera, named "kh3Kalman", whenever receiving a new odometry information by odom_Callback callback function; then getting the prediction posture of Khepera from the filter and publish to ROS network as "/odom_kalman_predict".
- Whenever receiving new measurements of corners, conducting the coherence test of new measurements by comparing the Mahalanobis distance with Threshold in extract_corner_Callback callback function. The measurement contain both distance and angle information of corners, but the current package is only tested with distance information. If there is only one valid corner after the coherence test, the posture of robot will be corrected by the new estimated posture and the program will publish a ROS message with topic "/odom_kalman_estimate".

Summary of this node can be described in *Figure 3.3*:

Figure 3.3: Extended Kalman Filter Localization node

Chapter 4

Experiments and Observations

4.1 Experiment configuration

In order to test the Kalman filter for localization in Khepera robot, we configured the experiment as follows:

- A clean white board with dimension 1 m$\times$1.4 m is used as running floor for the robot. The ”world” frame will be defined at the first corner of the board with x-axis along the length of the board.
- 4 corners of the board are marked by carton boards and bottles. The coordinate of these corners are known and defined in map1.xml file. They are used to help Khepera robot to correct its posture during movement.

Figure 4.1 displays the set up of the corner used in experiments:

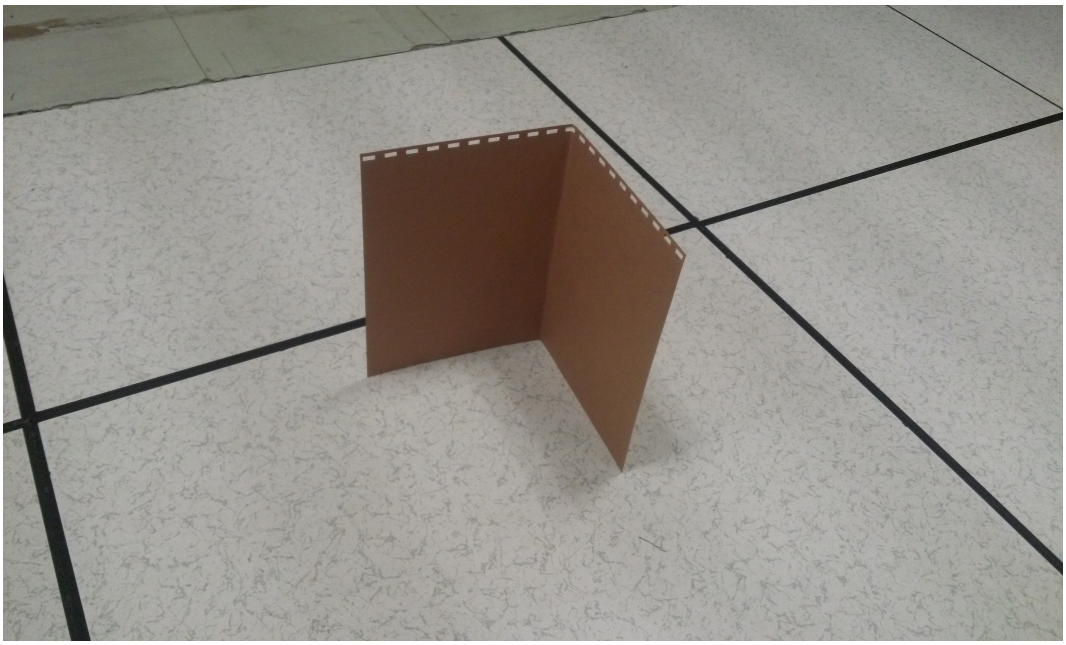

Figure 4.1: Corner set up for experiment

In addition, we used a launch file to start all packages needed for localization experiment: i.e, rosbag to record the content of all topic published to ROS network, playerros, rviz for visualization, line_extract, ekf_localize, planner to control the robot along a predefined trajectory (circle only in our tests), playerros_teleop to control Khepera from the keyboard. More detail of the launch file will be presented in the Appendix. In this launch file, the very important fixed frame ”world” at the first corner of the board is defined by using the package ”tf” as in 4.1:

Listing 4.1 Define fixed world frame in launch file

```
<node
        pkg="tf"
        type="static_transform_publisher"
        name="WORLD"
        args="0.48 0.805 0 0 0 -3.14 world odom 100">
</node>
```

The 3 first lines of 4.1 will define our fixed frame named WORLD. The last line of "args" will define the relation between the "world" frame and its "child frame", "odom" in this case, as (0.48, 0.805, 0.0) of translation and (0.0, 0.0, $-\pi$) of rotation with respect to the "world" frame. If the initial posture of Khepera is right at (x, y, θ) = (0.48, 0.805, $-\pi$) in the board, the "world" frame will be at the desired first corner. The number "100" at the end of the 4th line is the publishing period of this "world" frame in ms.

4.2 Experiments

Using the above configuration, we conducted some experiments as follows:

- Running in a circle with radius of 0.3 m from initial posture of (0.48, 0.805, $-\pi$) with good estimation of system noise and input noise
- Running in a circle with radius of 0.3 m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) with good estimation of system noise and input noise
- Running in a circle with radius of 0.3 m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) without laser signal
- Running in a circle with radius of 0.3 m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) with wrong estimation of system noise
- Running in a circle with radius of 0.3 m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) with wrong estimation of input noise

4.3 Results

The following figures present the results of all mentioned experiments respectively:

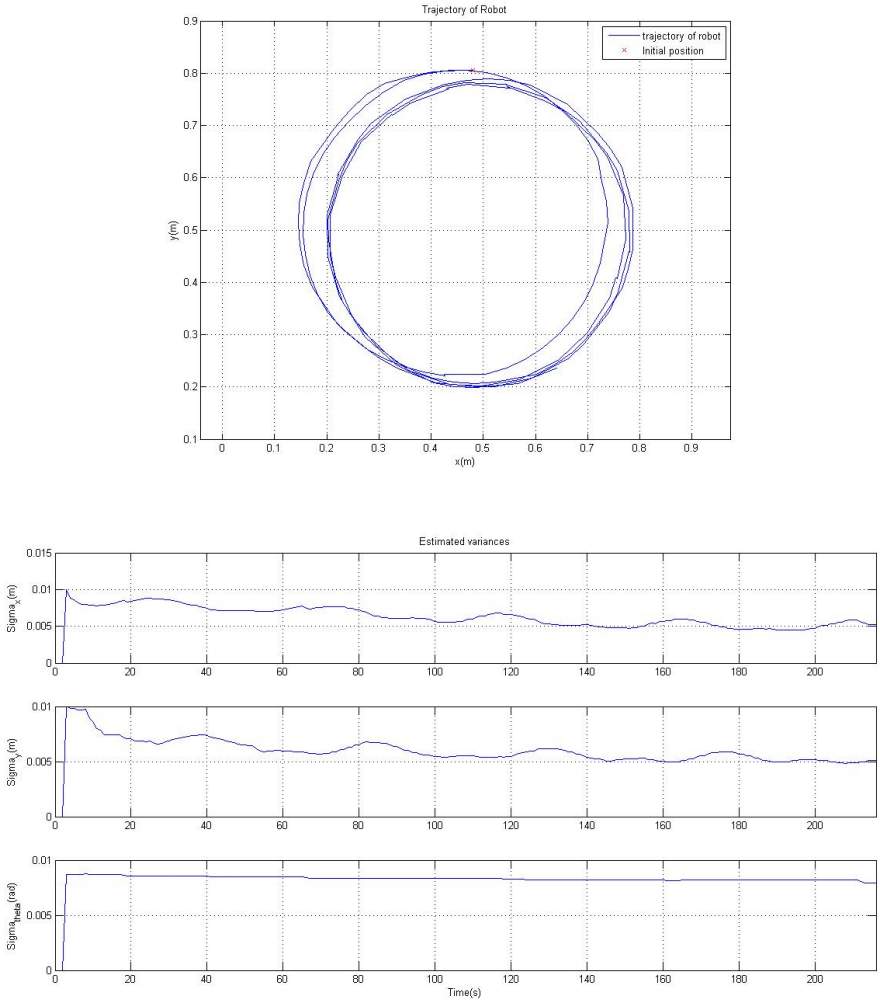

Figure 4.2: Running in a circle with radius of 0.3m from initial posture of (0.48, 0.805, $-\pi$) with good estimation of system noise and input noise: left:trajectory, right: Estimated variances

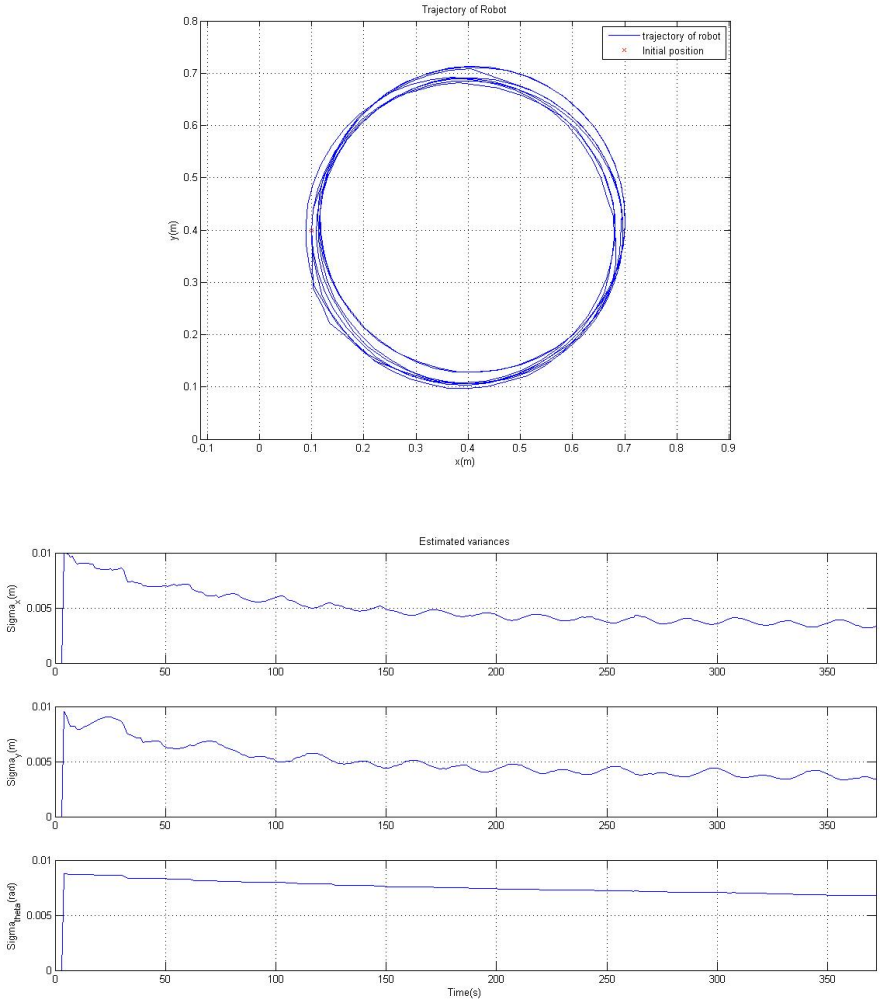

Figure 4.3: Running in a circle with radius of 0.3m from initial posture of $(0.1, 0.4, -\frac{\pi}{2})$ with good estimation of system noise and input noise: left:trajectory, right: Estimated variances

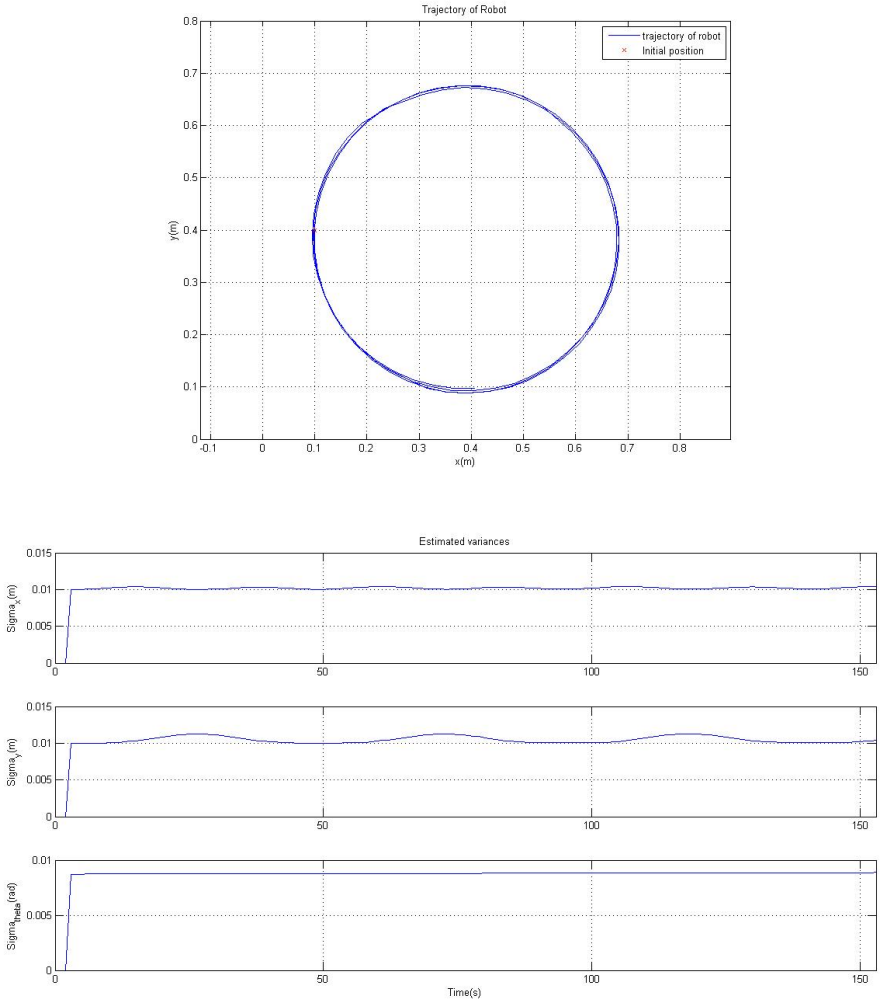

Figure 4.4: Running in a circle with radius of 0.3m from initial posture of (0.1, 0.4, $-\frac{\pi}{2}$) without laser signal

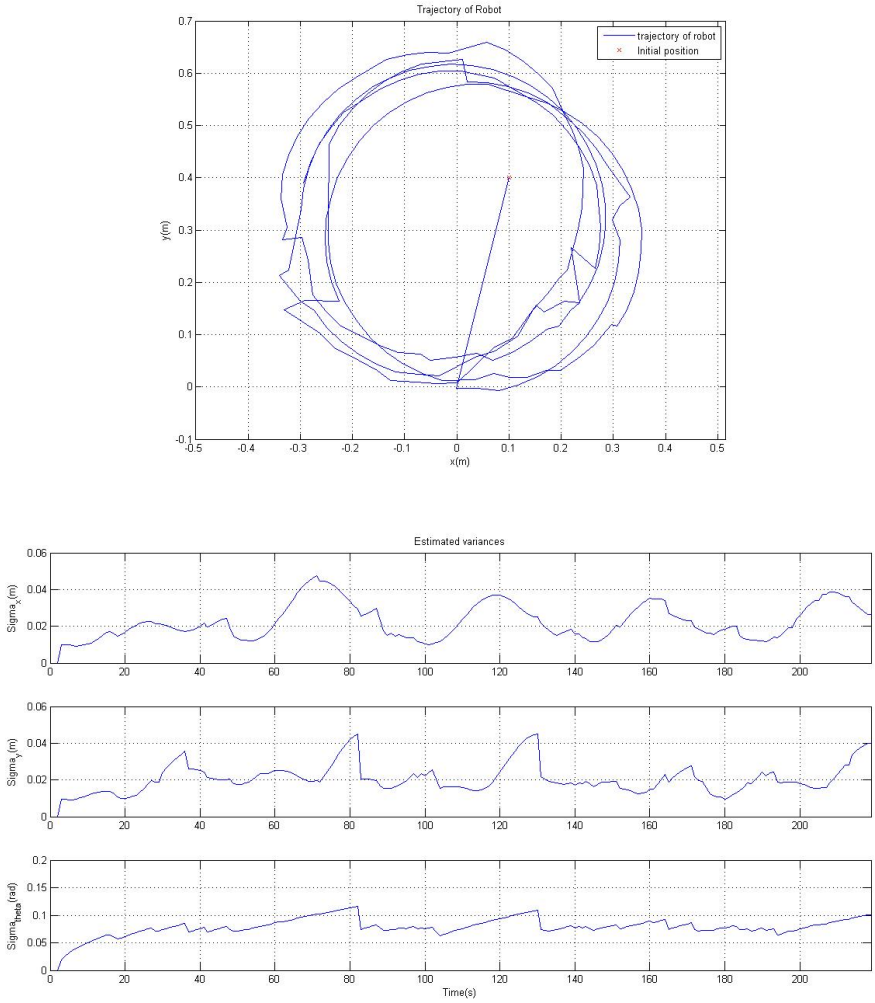

Figure 4.5: Running in a circle with radius of 0.3m from initial posture of $(0.1, 0.4, -\frac{\pi}{2})$ with wrong estimation of system noise: left:trajectory, right: Estimated variances

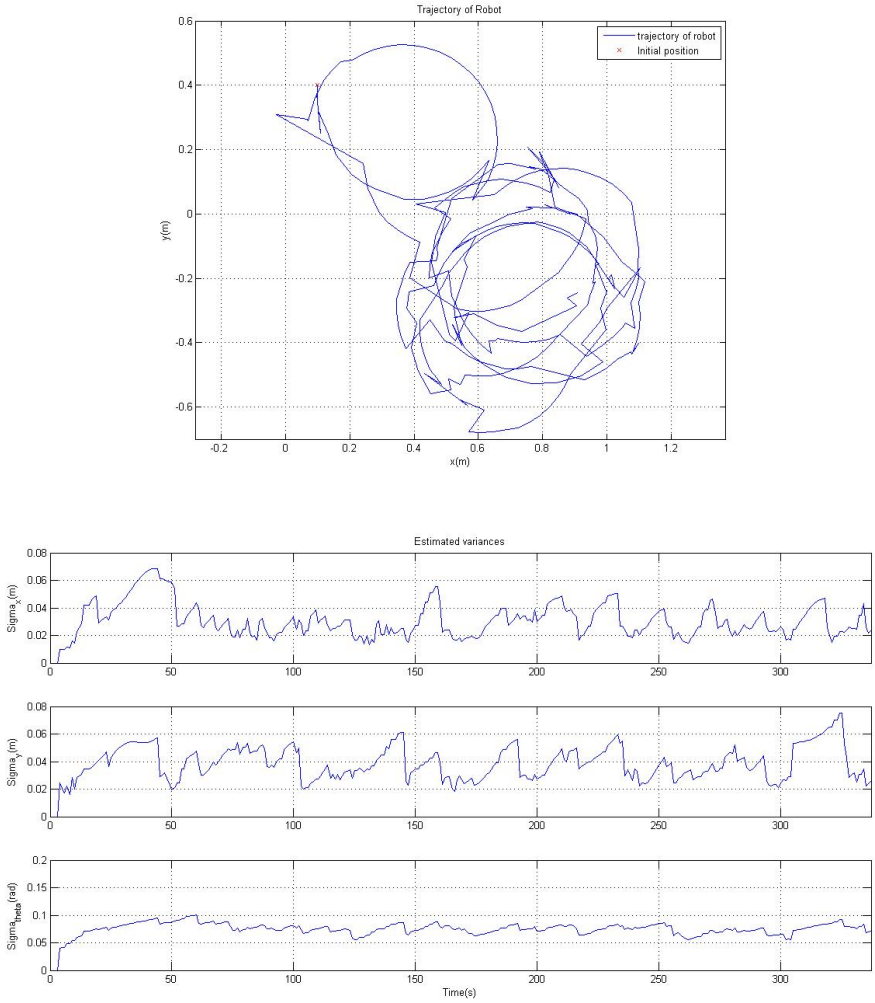

Figure 4.6: Running in a circle with radius of 0.3m from initial posture of $(0.1, 0.4, -\frac{\pi}{2})$ with wrong estimation of input noise: left:trajectory, right: Estimated variances

In the first two cases, as can be seen in *Figure 4.2* and *Figure 4.3* the estimations of trajectory of Khepera keep along circle with radius 0.3m as what the robot really did. There are some differences between circles over time because there were actually slipping and skidding during the movement of Khepera (the surface of the board has not enough friction). While the movement of Khepera without laser signal in the third case was similar to the second case but the estimation of trajectory has no difference over time (shown in the *Figure 4.4*) . These results prove that the implemented Kalman filter corrects the localization of the Khepera robot. The evolution of the estimated variance of posture in these three cases also supports the proof.

In the third case, there is very slow change in the variances of posture of robot while in two first cases there are gradual decrease of variances of posture over time. Normally, the covariance will increase over time due to the lack of correction of odometry, but in the third experiment the increase is very slow because we use the same good tuning parameters (zero system noise covariance and very small input noise covariance) as in first two experiments.

In the last two experiments, while the robot kept running in the circle as desired the estimation of robot pose are very bad as shown in the *Figure 4.5* and *Figure 4.6*. The reason is because of the

wrong estimation of system noise or input noise. There are very high fluctuations in estimated variance of posture (x,y,θ) in the both cases.

The following figures, *Figure 4.7* and *Figure 4.8* present the trajectory of Khepera on rviz tool of ROS:

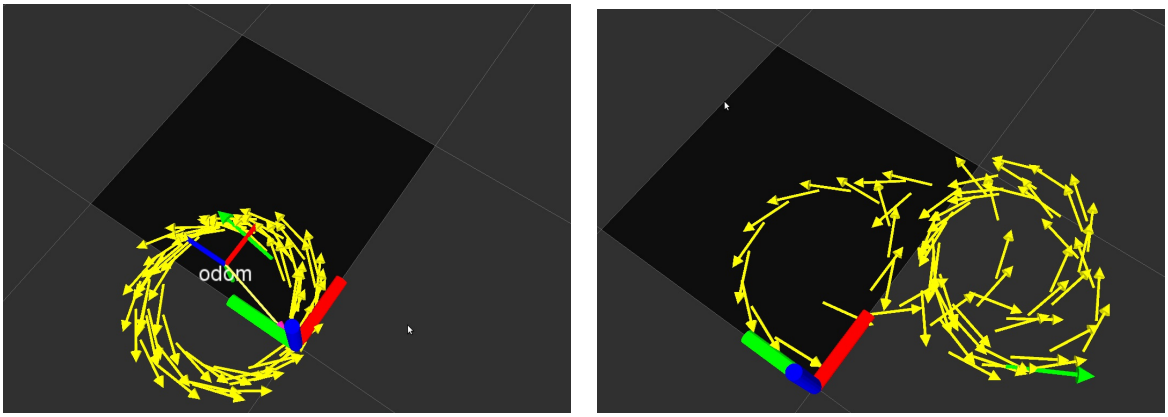

Figure 4.7: Trajectory of Khepera on rviz tool with wrong estimation: left:system noise, right: input noise

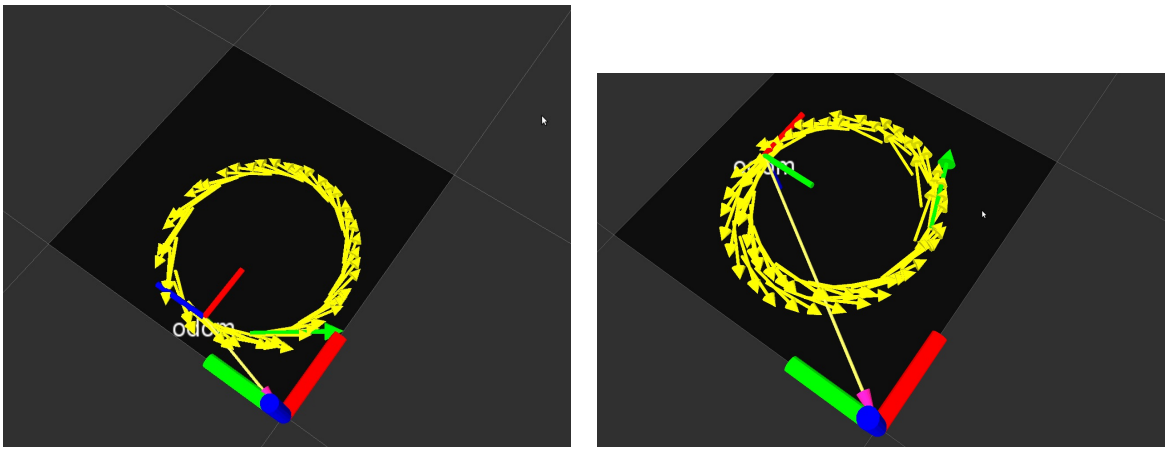

Figure 4.8: Trajectory of Khepera on rviz tool with good estimation: left:start at (0.1, 0.4, $-\frac{\pi}{2}$), right: (0.48, 0.805, $-\pi$)

Chapter 5

Conclusion

The results in Experiments and Results section show that our ROS applications can help Khepera to localize its posture in a known environment defined by map of feature corners. However the correction of posture by Kalman filter depends heavily on the precision of corners measurement by Laser Range Finder and laser data processing. Actually, the process of laser data still takes quite long time, leading to the slow and sometime not up-to-date posture correction, especially when Khepera moves fast enough. In addition, the laser data processing still can not work correctly in some posture of robot with respect to the corners, such as rendering 2 side of the corner as only 1 line thus Khepera can not detect that corner. The lack of correction of laser processing for all cases lower down the quality of the whole localization system.

In addition, the difference between the measured velocity by system and the real velocity of Khepera (mentioned in section of Player-ROS) actually produces very bad simulation of movement of robot, and this is a big problem for control Khepera movement if only use the odometry information. The presence of our localization with Kalman filter can help to solve the problem, at least in the case of controlling robot to move not too fast.

Future works With some drawbacks we recognize during conducting the project, we propose some future works as follows:

- Fixing the discrepancy between the measured velocity by system and the real velocity of Khepera and increasing the refresh rate of information from Player server in Khepera by checking the low level code written by K-team, with their support.
- Figuring out the reason of wrong process laser data to increase the correction of feature corners measurements.
- Applying the angle measurement to Kalman filter to compare both cases efficiency and also applying both measurement to Kalman filter to increase the correction of estimation.
- Applying the Kalman filter for the real map of full features of environment rather than only feature corners.
- SLAM

References

Nguyen, V., Martinelli, A., Tomatis, N., & Siegwart, R. (2005, Aug). *A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics.* doi: 10.1109/IROS.2005.1545234

Project, T. P. (2014). *The player project.* `http://playerstage.sourceforge.net/index.php?src=index`. ([Online; accessed 25-June-2014])

Roland Seigwart, I. R. N., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*. The MIT Press.

Teslic, L., Klancar, G., & Skrjanc, I. (2008, May). Kalman-filtering-based localization of a mobile robot with a lrf in a simulated 2d environment. In *Electrotechnical conference, 2008. melecon 2008. the 14th ieee mediterranean* (p. 316-322). doi: 10.1109/MELCON.2008.4618454

Appendix A

kh3_kalman_localize launch file

```
<launch>
  <param name="use_sim_time" value="false"/>

  <!-- Generate a fixed frame world at (0.0,0.0,0.0) and odom frame
       is translated ((0.48,0804,0.0) and -180 around z axis-->
  <node
        pkg="tf"
        type="static_transform_publisher"
        name="WORLD"
        args="0.48 0.805 0 0 0 -3.14 world odom 100">
  </node>

  <node
    pkg="rosbag"
    type="record"
    name="RECORD"
        args="-a -o /home/towardthesea/catkin_ws/src/bag/test">
  </node>

  <node
    pkg="playerros_teleop"
    type="playerros_teleop_node"
    name="Keyboard"
        output="screen">
  </node>

  <node
    pkg="playerros"
    type="playerros_node"
    name="Player_ROS"
        args="-h 192.168.1.150">
  </node>
<!--
  <node
    pkg="gmapping"
    type="slam_gmapping"
    name="Draw_Map"
        respawn="true">
        <param name="xmin" value="0"/>
        <param name="ymin" value="0"/>
        <param name="xmax" value="10"/>
        <param name="ymax" value="10"/>
        <param name="delta" value="0.05"/>
    <remap from="/scan" to="/base_scan"/>
  </node>
```

```
-->
  <node
          pkg="rviz"
          type="rviz"
          name="RVIZ"
          respawn="true">
  </node>

  <node
     pkg="line_extract"
     type="line_extract_node"
     name="LRF"
          respawn="true">
  </node>

  <node
     pkg="ekf_localize"
     type="ekf_localize_node"
     name="FILTER"
          respawn="true"
     args="-f $(find ekf_localize)/config1.xml -m $(find
         ekf_localize)/map2.xml"
          output="screen">
  </node>

  <node
     pkg="planner"
     type="planner_circle_run"
     name="CIRCLE"
          output = "screen">
  </node>

<!--
  <node
     pkg="kalman_localize"
     type="kalman_localize_node"
     name="FILTER"
          respawn="true"
     args="-f $(find kalman_localize)/config1.xml -m $(find
         kalman_localize)/map2.xml"
          output="screen">
  </node>
-->
 <!--
  <node
     pkg="randomwalk"
     type="randomwalk_node"
     name="Walk"
          respawn="true">
  </node> -->
```

```
</launch>
```

Appendix B

Section 1: Quick Start

Installing

- Download Player driver as following: (current version of Player at this time is 3.0.2)
  `http://playerstage.sourceforge.net/`
  Click on the "Download" button on left bar under the Sourceforge part.
  This library is important to link between ROS and Khepere through Player driver server running on Khepera

- Install Player driver as following: (this guide follows the "In-source Build" written in INSTALL file by Player creators).
  - Uncompress the Player downloaded file to Home folder in your Ubuntu machine and keep the name of the new uncompressed folder, for example "player-3.0.2" or you have to change the name of related path to Player driver in CMakeList.txt file of "playerros" package!!!
  - Open new Terminal and type:
    $ cd player-3.0.2
    $ cmake .
    $ make install
  - If errors occur during the build process, perform a verbose make:
    $ VERBOSE=1 make
  - If it still has problem, you should contact with Player project members, it is out of this Manual. But it works well with Ubuntu 12.04 both in pure installation and Virtual machine.

- Download and install ROS as following: (if you are using Ubuntu) `http://wiki.ros.org/hydro/Installation/Ubuntu`

- Configuring ROS Environment `http://wiki.ros.org/ROS/Tutorials/InstallingandConfi`

- Download a zip file from the following link and unzip to your /catkin_ws/src/ (if it is your source folder of ROS workspace)
  `https://drive.google.com/file/d/0B5uzFFH3HdnocGdxUkpubGh4MGs/edit?usp=sharing`
  There are 3 packages inside this zip file: playerros - communicating ROS with Khepera through Player driver (via Wifi), playerros_teleop - control robot with keyboard and randomwalk - simple controller for robot to walk randomly and avoid obstacles.

Now you are ready to run ROS with Khepera.

Running

This part is a guide to connect with Khepera through Wifi. Khepera will connect to a host spot named Taurus without password and has IP address: 192.168.1.150. Check the Section Wifi Connection Install guide for detail if you want to change this!

- Insert the Wifi card to Khepera. Make sure that you insert it before turn Khepera on. If NOT, you have to turn it off and repeat this step!
- Turn on the Khepera with battery or power cord.
- Wait for the blue and yellow LED light in Wifi card stop blinking, Khepera has finished connection to Wifi!
- Connect to Khepera through Wifi: open a terminal in your PC and type:
  $ ssh root@192.168.1.150 (or your new IP Address)
  then press Enter key for password (NO password)
- Make sure that Player was installed in Khepera. (Following part A (step 1 to 4) of Section of Player for Khepera III with KoreBot II Instructions 5)
- After installing Player in Khepera, continue:
  $ cd Setup
  $ ./laser.sh on
  $ player KheperaIII_urglaser_usb.cfg
  Now Khepera is waiting for command from PC
- Running ROS on your PC: open new terminal and go to launch folder in "randomwalk" package:
  $ cd /catkin_ws/src/randomwalk/launch
  $ roslaunch playerros.launch
  - If there is not Khepera at IP address (now is 192.168.1.150) set in "args" argument of launch file, there will be a red error in terminal. If there is Khepera at that IP address, robot will start to run.
  - There are 5 packages used in launch file:
    * playerros: publish laser scan as /base_scan topic, odometry as /odom topic, subscribe velocity command as /cmd_vel topic
    * playerros_teleop: publish /cmd_vel topic
    * randomwalk: publish /cmd_vel topic, subscribe /base_scan topic
    * gmapping to draw environment map (already installed with ROS)
    * rviz to show map, odometry , laser scan information (already installed with ROS)

Section 2: Player for Khepera III with KoreBot II
This section is part A of Instruction from K-team (can find in Khepera III CD-ROM).

Required:

- computer with linux 2.6 and Wifi access
- player 2.1.1 installed on the computer (`http://playerstage.sourceforge.net/`)
- Khepera 3 with Korebot II and Wifi (KoreWifi), Kernel 2.6
- from the binaries directory: (`http://ftp.k-team.com/KheperaIII/player_stage/korebotII/binaries/`)
  - Khepera 3 player driver: KheperaIII.so
  - Khepera 3 player driver configuration file:
    KheperaIII.cfg: to work without Hokuyo Laser Range Finder
    KheperaIII_urglaser_usb.cfg: to work with Hokuyo Laser Range Finder
  - Player needed libraries:
    libltdl3_1.5.10-r3_armv5te.ipk
    libstdc++6_4.1.2-r10_armv5te.ipk
  - laser.sh script

Establish a network connection with the Khepera 3/Korebot 2 see Korebot 2 user manual, chapter "5.2.2 Using a Wireless compact flash card":

`http://ftp.k-team.com/KorebotII/UserManual/`

Copy and install the 3 following packages on the Korebot:

- command for copying:
  $ scp PACKAGE_NAME root@KHEPERA_IP_ADDRESS:/home/root
- installation procedure :
  $ ipkg install PACKAGE_NAME
- packages:
  - c++ standard library: libstdc++6_4.1.2-r10_armv5te.ipk
  - libtool: libltdl3_1.5.10-r3_armv5te.ipk
  - player server: player_2.1.1-r0_armv5te.ipk
- If there is not enough free space, delete each package after installation.

Copy KheperaIII.so and KheperaIII.cfg to the Korebot:
Using Serial connection 5 or Wifi Connection **??**.

Copy the laser.sh script to the Korebot:

Make it executable and run it with the commands:
$ chmod +x laser.sh
$ ./laser.sh on

On the KoreBot, launch the server:

```
$ player KheperaIII.cfg
```

On the computer in a terminal, export the library path and launch the viewer:
(player 2.1.1 must be installed on the computer):

```
$ export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH

$ playerv -h IP_ADDRESS_OF_THE_ROBOT –ir:0 –position2d:0 –sonar:0 –power:0
```

Drive the Robot:

- either with the playerv interface :
  - Go to ”Devices/Position2d” and select ”Command”
  - A red cross appears on the robot.
  - You can move it to drive the robot.
- or with ”playerjoy”:
  - the keyboard:
    ```
    $ playerjoy IP_ADDRESS_OF_THE_ROBOT
    ```
  - a standard joystick connected:
    ```
    $ playerjoy -speed 1.0 -turnspeed 90 -dev /dev/input/js0 IP_ADDRESS_OF_THE_ROBOT
    ```

Section 3: Serial Connection with Khepera III

Using minicom:

- Install minicom
  ```
  $ sudo app-get install minicom
  ```
- Setup minicom
  ```
  $ minicom -s
  ```
- Change the serial port which connect to Khepera
- Using administration to setup minicom for the first time and save configuration for next uses
  ```
  $ sudo minicom -s
  ```
- Get access permission to serial ports if meet ”permission denied error” when try to connect to Khepera

  - check connection
    $ ls -l /dev/ttyUSB*
  - verify if the user belongs to the dialout group
    $ id -Gn ¡username¿
  - add user to the ”dialout” supplementary group
    $ sudo usermod -a -G dialout ¡username¿

- Log out and log in before changes take effect. Test again by:
  $ id - Gn ¡username¿
- Turn on Khepera and wait for booting.

Using Khepera:

Note: open 1 terminal and connect with robot through minicom; open another terminal to compile program

- First step
  - Note: Connect with minicom: configuration (pages 9, KoreBot Manual)
    * 115200 Bps
    * 8 data bits
    * No parity
    * 2 stop bits (not 1)
  - Follow 4.2 (KoreBot Manual), 4.3
  - Install Light toolchain (4.4)
- Build your program
  - Go to folder of program
  - Using command: (page 37, Khepera II manual) to build your program
    $ source ../../env.sh (numbers of ../ depends on path of program folder
    $ arm-angstrom-linux-gnueabi-gcc ¡filename.c¿ -o ¡output¿ -I $LIBKOREBOT_ROOT/build-korebot-2.6/include -L $LIBKOREBOT_ROOT/build-korebot-2.6/lib -lkorebot
- Copy to Khepera/KoreBot (page 45, KoreBot Manual)
  Note: Using minicom
  - In the Minicom console, hold the keys ”Ctrl + a” and press ”s” and select ”Z-Modem”.
  - Select the file you would like to upload to the Korebot (navigate with the arrows keys, 2x ”spacebar” to change directory and ”spacebar” to select the file).
  - Select [Okay] to send it.

*** Can test with kh3test.c in /development_k2_v1.0/libkorebot-1.19-kb1/src

Wifi Connection for Khepera III with KoreBot II:

Follow the Section 5.2.2 of KoreBot II Manual

5.0.1 Set the wifi host spot without password

- Insert a Wireless compact flash card in the Korebot before it is turned on
- Load the module by typing:
  $ modprobe pxa2xx_cs
  You may load the Wifi module automatically by adding pxa2xx_cs in the file
  $ /etc/modules
  You can use the following command echo to add the module name to the file:
  $ echo pxa2xx_cs ¿¿/etc/modules
- Configure the wireless network:
  Without any encryption for security:
  Modify the file /etc/network/interfaces with your wireless network settings with vi editor (see chapter 5.2.7 "Using vi text file editor"=¿ Note for vi text editor):

```
/***************** /etc/network/interfaces **********/
# The loopback interface
#
auto lo
iface lo inet loopback
#
# Wireless interfaces
#
auto eth0
#iface eth0 inet dhcp
iface eth0 inet static
wireless\_mode managed
wireless\_essid YOUR\_SSID\_OF\_NETWORK (name of the
    Wifi host spot)
address YOUR\_IP\_ADDRESS (IP Address of Robot used to
     control it later)
netmask YOUR\_NETMASK (can be 255.255.255.0)
gateway YOUR\_GATEWAY\_IP (can be the same as IP
    address)
/*********************************************/
```

5.0.2 Using:

Restart robot
  $ reboot
Connect to robot through wifi
  $ ssh root@ROBOT_IP_ADDRESS (set above)

Tried with WEP_TKIT but not succeed yet!!!

```
+++Note for "vi text editor
    ":+++++++++++++++++++++++++++++++
```

```
        Arrow key for moving cursor
        i                insert function
        a                append function
        ESC              stop insert/append function
        ZZ               save and quit
        :q!              quit without saving
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```