Applying Kalman filter to Robot networks

Gu Peizhen (Alan)

University of California, San Diego

Summer 2014

0 ABSTRACT

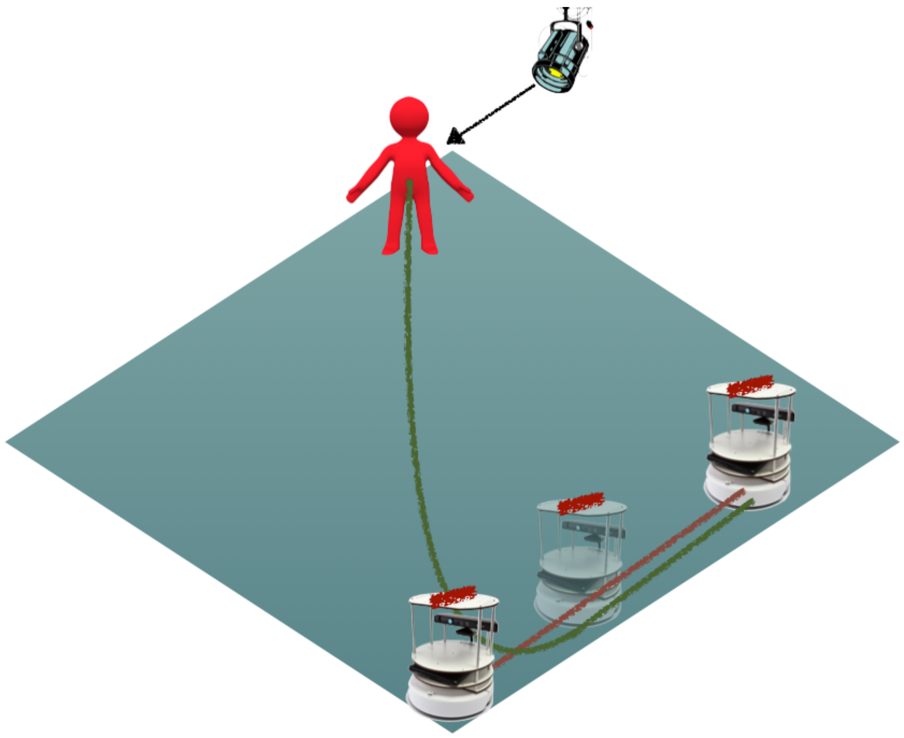
This paper shows my work of applying Kalman filter to agents in the lab. Based on Robot Operating System (ROS), Kalman filter can provide more precise estimations about the location of agents. The Kalman filter node is connected with two launch files: one can be used for robot deployment. The other one combines with keyboard controller (teleop), and can be used for parameter adjustment.

1 BIG PICTURE

This part mainly contains the objectives of the agent localization project and my own subjects.

1.1 Localization of Mobile Robots

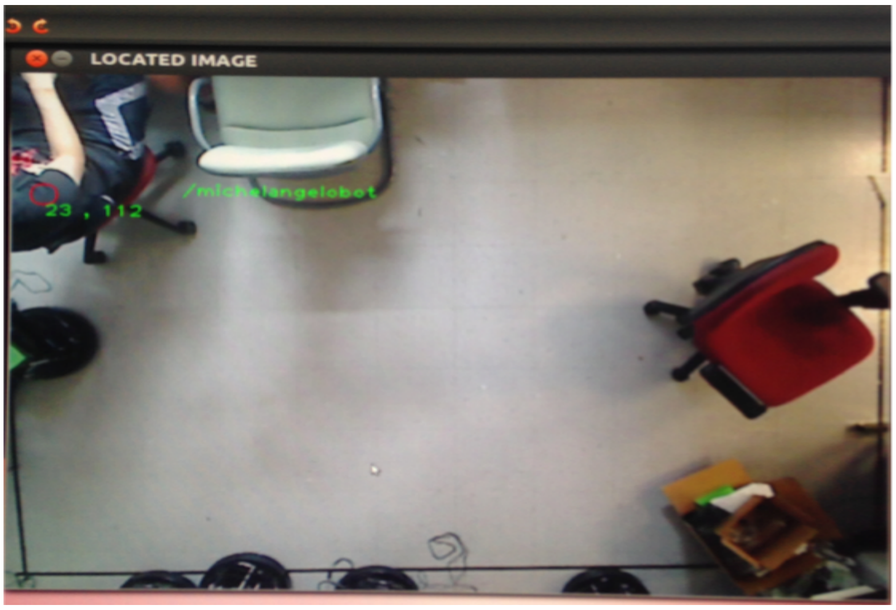
Until now, we have been using colored tags on the top of the agents to perform localization tasks. However, as light condition may not be that preferable at some specific time or spot, information collected by the camera may not be as reliable. Let's see the example below:



Assume that an agent with a red tag on the top of it is moving forward, and following the red line. Due to some specific changes of the light condition, the robot disappeared for a second in the camera.

Meanwhile, there is a man in red (or some other red stuff) in court. The camera, busy searching for the red tag, can easily recognize the red stuff as the agent. Therefore, in view of the system, our robot is moving in the green line.

We may find similar situations in our daily work:



In the picture, the camera took my T-shirt as one of its agents. Mistakes like this can significantly affect the robustness of the system.

1.2 Applying Kalman filter

My work aims to build a node of Kalman filter for agents, which can provide stable and precise localization information by combining both the camera data and the system prediction values. The filter was written in C++ and can be operated on Robot Operating System (ROS). Details about the node will be discussed later.

2 PERSONAL CONTRIBUTIONS

My main contribution in this project is to develop a Kalman filter and to substitute estimation from Kalman filter for the original topic "amcl_pose" to be the source of position information of the topic "/all_positions". In other words, the system does not use the camera data directly. Camera information is instead sent to Kalman filter at first, the filter then generates the final estimation, and forms the position information of the agents.

What's more, by introducing the keyboard controller (teleop), I wrote a launch file named "kalmanfilter.launch". With this file, we can use a keyboard to control robots, which allows us to see how well the filter works. Through this step we can adjust some of the parameters in the filter node to enhance its performance.

3 PRELIMINARIES

In this section, some basic topics crucial for understanding the Kalman filter are discussed.

3.1 Extended Kalman filter (EKF)

Kalman filter can be divided into two parts: time update part and measurement update part, and it works in the following patterns:

A state space model can be described like this:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}$$
$$z_k = Hx_k + v_k$$

Where x is the state space matrix of the robot, z is the system measurement. Matrix w and v contain error information while u is the system input.

However, the model of our robot is not a linear one, so we need to introduce the extended Kalman filter (EKF). The EKF was designed for the models like this:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$
$$z_k = h(x_k, v_k)$$

This model needs to be linearized at first before we introduce the filter:

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0)$$

$$B_{[i,j]} = \frac{\partial f_{[i]}}{\partial u_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0)$$

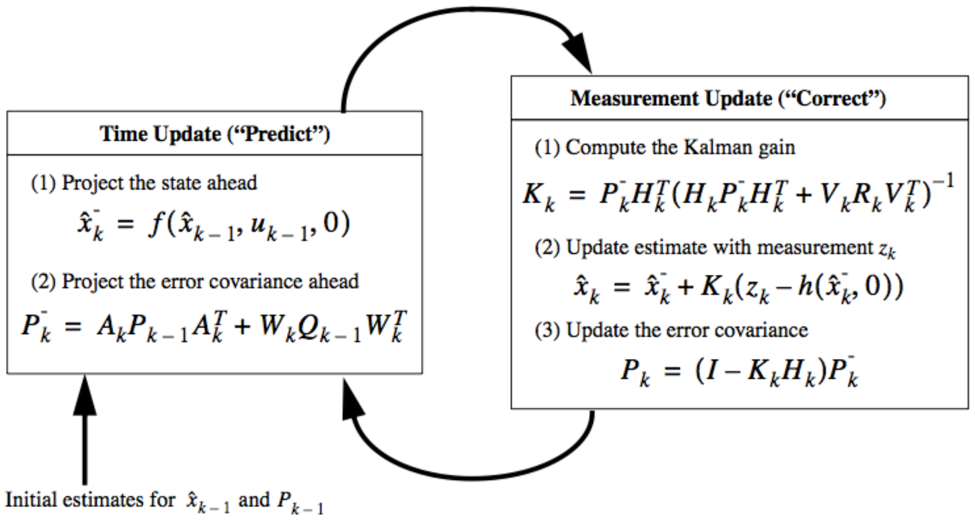
$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\hat{x}_k^{-}, 0) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\hat{x}_k^{-}, 0) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The feature of our system determines that H, W and V should be identity matrix.

EKF works in the following patterns:


Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0)$
(2) Project the error covariance ahead
$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0))$
(3) Update the error covariance
$P_k = (I - K_k H_k) P_k^-$
Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

More details will be discussed in "METHODOLOGY" part.

4 METHODOLOGY

In this section, details of the filter will be discussed, including the mathematical model, interface design, code design and introduction for some important parameters. The code and notes of Kalman filter can be found under:

lab/in-progress/deployment/turtlebot_deployment/src/Kalmanfilter.

cpp.

4.1 Mathematical model

For more details, see *An introduction to the Kalman filter*. (Welch G, Bishop G.1995)[1]

The state transition model of agents is simple. We choose horizontal ordinate (x), longitudinal coordinate (y) and direction angle (θ) as the state space variables. The system has two kinds of input: linear velocity (v) and angular velocity (ω). State transition model can be described as follows:

$$
\begin{aligned}
x_k &= x_{k-1} + T \cdot v \cdot \cos(\theta) + w_1 \\
y_k &= y_{k-1} + T \cdot v \cdot \sin(\theta) + w_2 \\
\theta_k &= \theta_{k-1} + T \cdot \omega + w_3
\end{aligned}
$$

w is the interference of the system, while T represents the time period, which is determined by how quickly the Kalman filter node spin. x_{k-1} and x_k represent the x coordinate at present and coordinate a time period later respectively. As we can see, this system is not a linear one, so we need to follow EKF to linearize it into this form:

$$
\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = A \cdot \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + B \cdot \begin{bmatrix} v_{k-1} \\ \omega_{k-1} \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}
$$

Where:

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0) = \begin{pmatrix} 1 & 0 & -T \cdot v \cdot \sin(\theta_{k-1}) \\ 0 & 1 & T \cdot v \cdot \cos(\theta_{k-1}) \\ 0 & 0 & 1 \end{pmatrix}$$

$$B_{[i,j]} = \frac{\partial f_{[i]}}{\partial u_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0) = \begin{pmatrix} \cos\theta_{k-1} & 0 \\ \sin\theta_{k-1} & 0 \\ 0 & T \end{pmatrix}$$

$$H_{[i,j]=} \frac{\partial h_{[i]}}{\partial x_{[j]}}(\hat{x}_k^{-}, 0) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$p(w) \sim N(0, Q)$$
$$p(v) \sim N(0, R)$$

Using this model, the filter node was built in the following pattern:

Eq.1: $\hat{x}_k^{-} = f(\hat{x}_{k-1}, u_{k-1}, 0)$

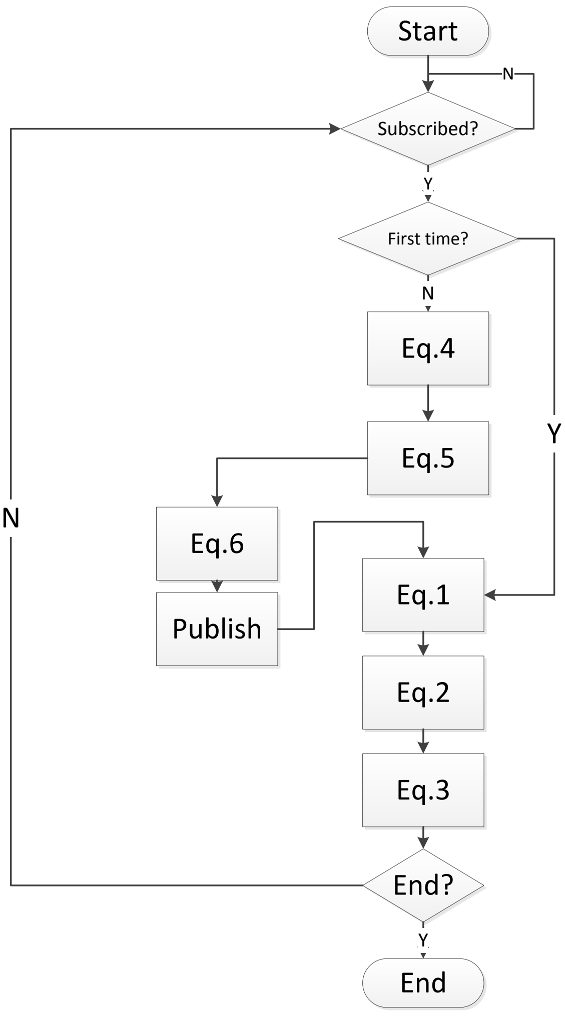
Eq.2: $P_k^{-} = A_k \cdot P_{k-1} \cdot A_K^{T} + W_k \cdot Q_{k-1} \cdot W_k^{T}$

Eq.3: $K_k = P_k^{-} \cdot H_k^{T} \cdot (H_k \cdot P_k^{-} \cdot H_k^{T} + V_k \cdot R_k \cdot V_k^{T})^{-1}$

Eq.4: $\hat{x}_k = \hat{x}_k^{-} + K_k(z_k - h(\hat{x}_k^{-}, 0))$

Eq.5: $P_k = (I - K_k \cdot H_k) \cdot P_k^{-}$

Eq.6: $\hat{x}_{k-1} = \hat{x}_k \quad , \quad P_{k-1} = P_k$



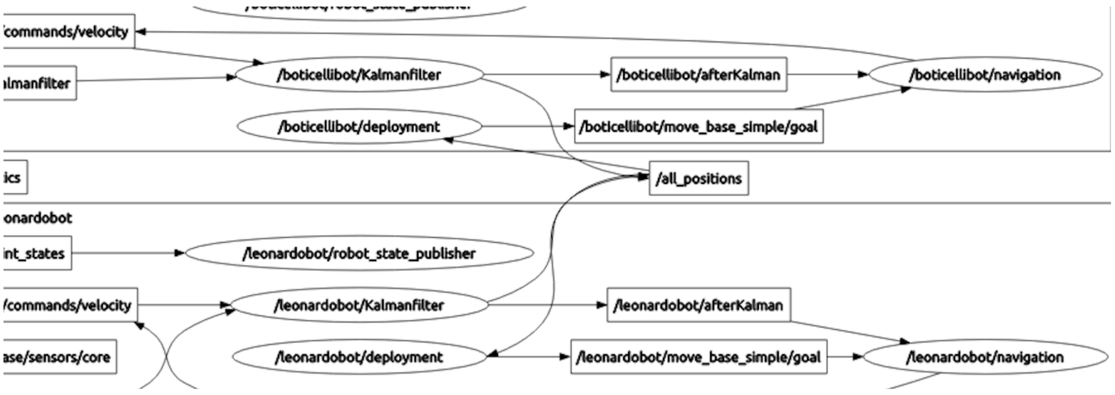
Set the initial value of x and P to be:

$$x_0 = \begin{bmatrix} 1.628 \\ 1.647 \\ 0 \end{bmatrix} \qquad P_0 = \begin{pmatrix} 9 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

which means the system assumes the initial position of x is right in the center of the court and it is not sure about this estimation. This procedure will make the result to converge to the real value more quickly.

4.2 INTERFACE DESIGN

The figure below shows part of the rqt_graph of the whole system:



In the picture, Kalman filter node is called **"/*Robotname*/Kalmanfilter"**, this node:

<u>Subscribes from topics:</u>

"/Robotname/mobile_base/commands/velocity "(contains velocity and angular velocity);
"/Robotname/toKalmanfilter"(contains position information from camera)

<u>Advertise to the topics:</u>

"/all_positions"(contains all position information about the agents in the system)
"/Robotname/afterKalman"(contains the position information of the single agent itself)

As a result, the interface in the node was designed as follow:

```
pos_sub_=nh_.subscribe<turtlebot_deployment::PoseWithName>("toKalmanfilter",
10, &Kalmanfilter::posecallback, this);
Ipt_sub_=nh_.subscribe<geometry_msgs::Twist>("mobile_base/commands/velocity",10
,&Kalmanfilter::iptcallback, this);
gl_pub_=nh_.advertise<turtlebot_deployment::PoseWithName>("/all_positions", 10,
true);
sf_pub_=nh_.advertise<turtlebot_deployment::PoseWithName>("afterKalman",10,true
);
```

4.3 CODE DESIGN

After finishing the design of the interface, I found that there is a deadlock in this structure:

At the beginning of the deployment, there is no message advertised on the topic “velocity”. As a result, the node Kalmanfilter will not be able to publish anything to the topic “all_positions” and “afterKalman”. Voronoi generator cannot generate the Voronoi cell, this will in turn result in the silence of “velocity” again.

To solve this problem, code was added in the Kalman filter (line 261-271) like this:

```
if(counts==0){ geometry_msgs::Pose goalPose;
      goalPose.position.x = X1[0][0];
      goalPose.position.y = X1[1][0];
      goalPose.orientation =tf::createQuaternionMsgFromYaw(X1[2][0]);
       newPose_->pose = goalPose;
       gl_pub_.publish(newPose_);
       sf_pub_.publish(newPose_);}
       counts++;
    }
```

The filter will at first publish the raw estimation of the state values at first, to serve as a trigger of the whole system. Then the system will spin it self and this part of code will never be executed again.

4.4 INTRODUCTION FOR SOME IMPORTANT PARAMETERS

4.4.1 Time Period T

The time period of the system T is set to be 2, which requires the system to publish information at a frequency of 0.5 Hz, the code can be found as follows (line 286-295):

```
int main(int argc, char **argv)
{
   ros::init(argc, argv, "Kalmanfilter");
   Kalmanfilter kalmanfilter;
   ros::Rate loop_rate(0.5);
   while (ros::ok()) {
       ros::spinOnce();
       loop_rate.sleep();
}
```

4.4.2 Matrix P

Parameters in Matrix P indicate how much the information of the camera will be considered by the system. However, I found that after the agent had operated for a long time, especially after the robot moved through areas where its image may not be captured by the camera, the parameter in P could become abnormal and that may eventually caused the estimation incorrect.

As a result, I added the following codes into the system (line 203-209):

```
for (i=0;i<=2;i++){
          for (j=0;j<=2;j++){
            if( P1[i][j]<=0.1){P1[i][j]=0.1;}
 if( P1[i][j]>=5){P1[i][j]=0.1;}
std::cout<<"P1="<<P1[i][j]<<"\n";
          }
       }
```

To make the filter perform better, we can adjust the upper and lower level of matrix P by launching Kalmanfilter.launch (procedures will be introduced later).

5 TIPS

This part contains useful information about how to launch the Kalman filter.

5.1 DEPLOMENT MANUAL

5.1.1 BASIC SETUP

(See P26-28 of the report from Rokus[2] for more details):

1. Updating the software

```
$    sudo apt-get update

$    sudo apt-get dist-upgrade

$    sudo reboot
```

2. Network setup
3. Environment setup
4. Launching the camera node:

```
$    roslaunch blueBot localization demo multiple.launch
```

5. Starting a map-server:

```
$    roslaunch turtlebot deployment map server.launch
```

5.1.2 DEPLOYING THE ROBOT

1. Launch the deployment launch file

```
$    roslaunch turtlebot deployment deploy robot.launch robot:=[robot name]
```

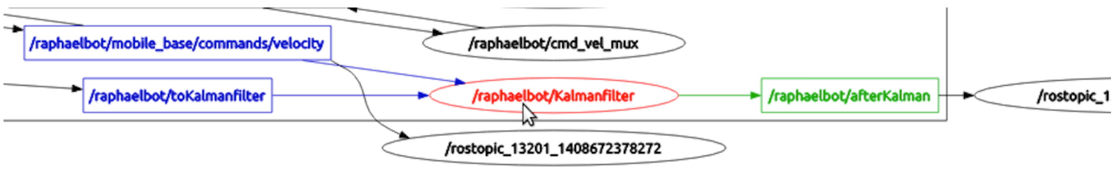
2. View the status of the filter

```
$    rostopic echo /raphaelbot/afterKalman
$    rostopic echo /raphaelbot/amcl_pose
```

Through "rostopic echo" we can see messages advertised on a specific topic The position estimations generated by the Kalman filter are published on the topic “afterKalman”, while the positions generated by the camera are published on the topic “amcl_pose”. Through this command we can see how well the Kalman filter works directly.

5.2 LAUNCH KALMANFILTER SEPARATELY

I have written a Kalman filter launch file without the node “simple_deployment” and “first_turn_then_move”. The rqt_graph is as follows:


/raphaelbot/mobile_base/commands/velocity
/raphaelbot/cmd_vel_mux
/raphaelbot/toKalmanFilter
/raphaelbot/Kalmanfilter
/raphaelbot/afterKalman
/rostopic_1
/rostopic_13201_1408672378272

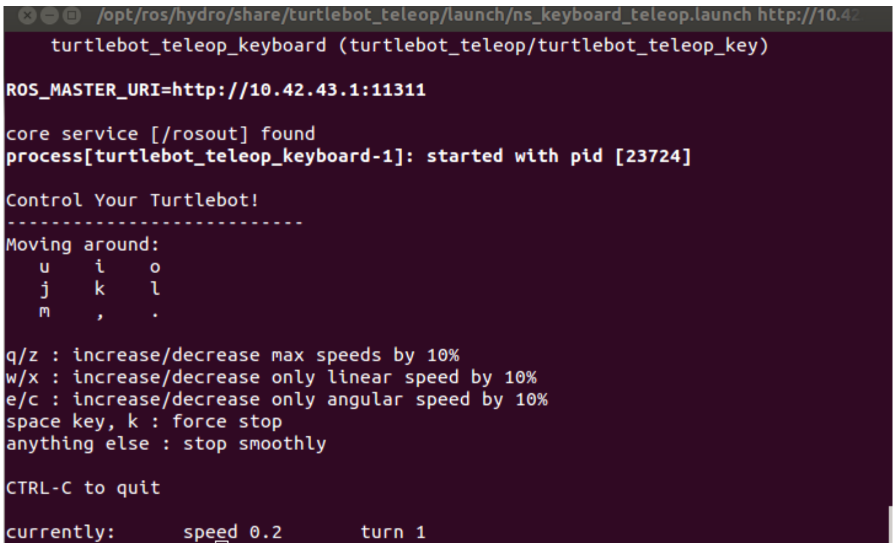
We can launch this file using command:

```
$     roslaunch kalmanfilter Kalmanfilter.launch   robot:=[robot name]
```

After launching this launch file, agents can only be commanded by keyboard:

```
$     roslaunch turtlebot_teleop ns_keyboard_teleop.launch
```

When the step is finished, it appears on the window:



```
/opt/ros/hydro/share/turtlebot_teleop/launch/ns_keyboard_teleop.launch http://10.42
    turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)

ROS_MASTER_URI=http://10.42.43.1:11311

core service [/rosout] found
process[turtlebot_teleop_keyboard-1]: started with pid [23724]

Control Your Turtlebot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1
```

Press key “J, I, L” on the keyboard, and we can control the turtlebot to move around.

This launch file is connected with “Kalmanfilter.cpp”, and you can find and change this node under “robot_deployment/src” instead of “kalmanfilter/src”.

6 PITFALLS

As is mentioned before, P matrix may become abnormal after the agent has operated for a long time. The reason needs further discussion.

Our solution by now is to set an upper and lower margin value to keep the parameters from getting too big or too small.

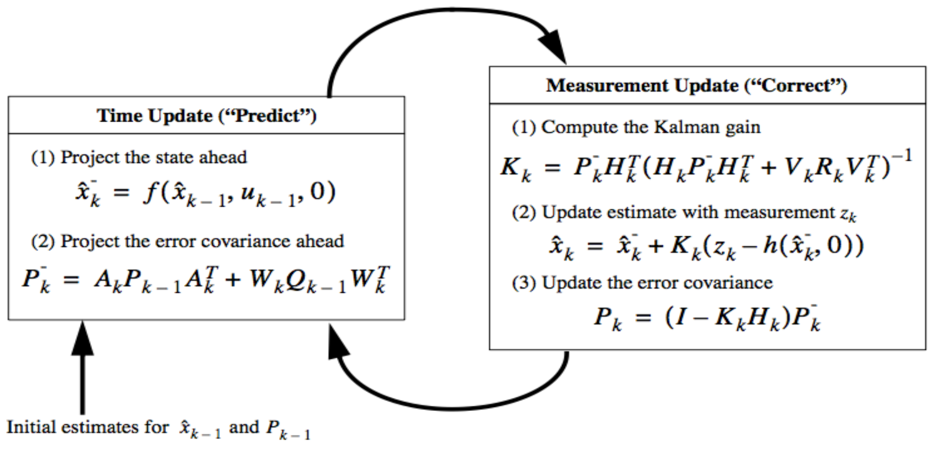
7 CONCUSIONS & FUTURE DIRECTIONS

Evidence shows that Kalman filter could provide more stable and precise position information than the camera. When the camera calibration doesn’t work, Kalman filter can helps the system to keep stable.

We are using colored tags to locate our robots. However, this can be vulnerable to the changes of light conditions. Using shapes in black and white may solve the problem. With the help of OpenCV, we can draw some specific shapes on tags, like triangle or rectangle, and print them in black. The camera can capture their margin lines in order to derive the location and direction information.

8 NOTES FOR VARIABLES

This part shows the meanings of the variables in Kalmanfilter.cpp.



Time Update (“Predict”)
(1) Project the state ahead
$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0)$
(2) Project the error covariance ahead
$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
Measurement Update (“Correct”)
(1) Compute the Kalman gain
$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0))$
(3) Update the error covariance
$P_k = (I - K_k H_k) P_k^-$
Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

T:	Time Period
A, H, Q, R, K:	Same as the A, H, Q, R, K in the model
X0:	$\hat{x}_{k-1}$
X1:	$\hat{x}_k^-$
X2:	$\hat{x}_k$
P0:	P_{k-1}
P1:	P_k^-
P2:	P_k
AT:	A^T

9 REFERENCES

[1] Welch G, Bishop G. An introduction to the Kalman filter[J]. 1995.

[2] R.A. Ottervanger, Implementation of a distributed algorithm for coverage control, Internship report, 2014