

Programming Assignment 2

Question 1:

Goal: Get pose estimate.

- a) Implement a Kalman Filter and write a node within a ROS package that subscribes to `cmd_vel` and `scan` to process the messages for propagation and update. You can assume the same scenario shown as an example in class: the robot moving on a line, and having the wall, i.e., the laser scan measurement in front of the robot, as the landmark. The node should publish a message of type `PoseWithCovarianceStamped` with the current estimate and covariance matrix. Note that you need to initialize some of the parameters of the Kalman Filter, including the noise on sensor and motion.
- b) For graduate students: Publish also a transformation using `tf`, where the global reference frame should be called `odom_kf` and goes to `base_footprint`.
- c) The Kalman Filter should subscribe to `initialpose` (`geometry_msgs/PoseWithCovarianceStamped` message type) so that the initial state can be provided.

Solution:

a) In the first task, command (`roslaunch FILE_MANE`) has been run to publish some sensing information that has been recorded in the previous assignment. Also, a ROS node has been implemented that subscribes to some of these recorded topics including (`/scan`, `/pose`, and `/initialpose`). Two kalman filters have been implemented in this node to estimate the distance that has been traveled by the robot since the robot starts moving. The first filter depends on `/cmd_vel` topic to calculate the robot's state through multiplying the robot's velocity by the traveling time, while the second filter depends on subscribing to the `/pose` topic to compute the robot's state. The Kalman Filter equations that have been used to estimate the robot's state are as follows:

Propagation (motion model):

$$\begin{aligned}\hat{x}_{t+1/t} &= F_t \hat{x}_{t/t} + B_t u_t & - \text{State estimate is updated from system dynamics} \\ P_{t+1/t} &= F_t P_{t/t} F_t^T + G_t Q_t G_t^T & - \text{Uncertainty estimate GROWS}\end{aligned}$$

Update (sensor model):

$$\begin{aligned}\hat{z}_{t+1} &= H_{t+1} \hat{x}_{t+1/t} & - \text{Compute expected value of sensor reading} \\ r_{t+1} &= z_{t+1} - \hat{z}_{t+1} & - \text{Compute the difference between expected and "true"} \\ S_{t+1} &= H_{t+1} P_{t+1/t} H_{t+1}^T + R_{t+1} & - \text{Compute covariance of sensor reading} \\ K_{t+1} &= P_{t+1/t} H_{t+1}^T S_{t+1}^{-1} & - \text{Compute the Kalman Gain (how much to correct est.)} \\ \hat{x}_{t+1/t+1} &= \hat{x}_{t+1/t} + K_{t+1} r_{t+1} & - \text{Multiply residual times gain to correct state estimate} \\ P_{t+1/t+1} &= P_{t+1/t} - P_{t+1/t} H_{t+1}^T S_{t+1}^{-1} H_{t+1} P_{t+1/t} & - \text{Uncertainty estimate SHRINKS}\end{aligned}$$

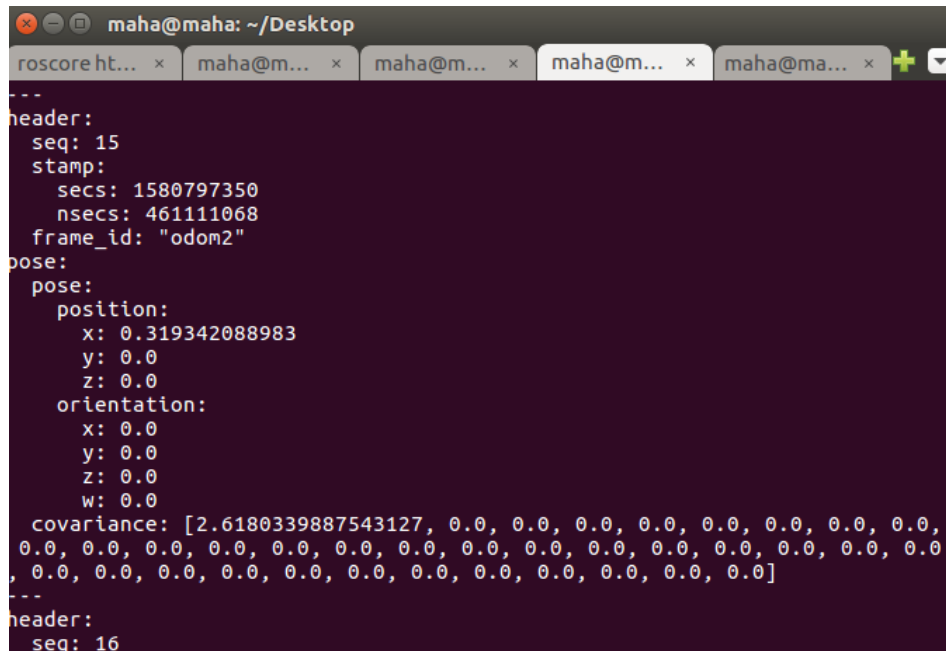
Illustration a: Kalman Filter's equation

This filter composed of two steps:

1- The propagation step: in which the state is estimated from the action which is the velocity here. The uncertainty also is estimated.

2- The update step: in which the uncertainty and the robot's state are estimated using the robot's observation which is here the LIDAR information (the distance between the robot and the wall in front of the robot).

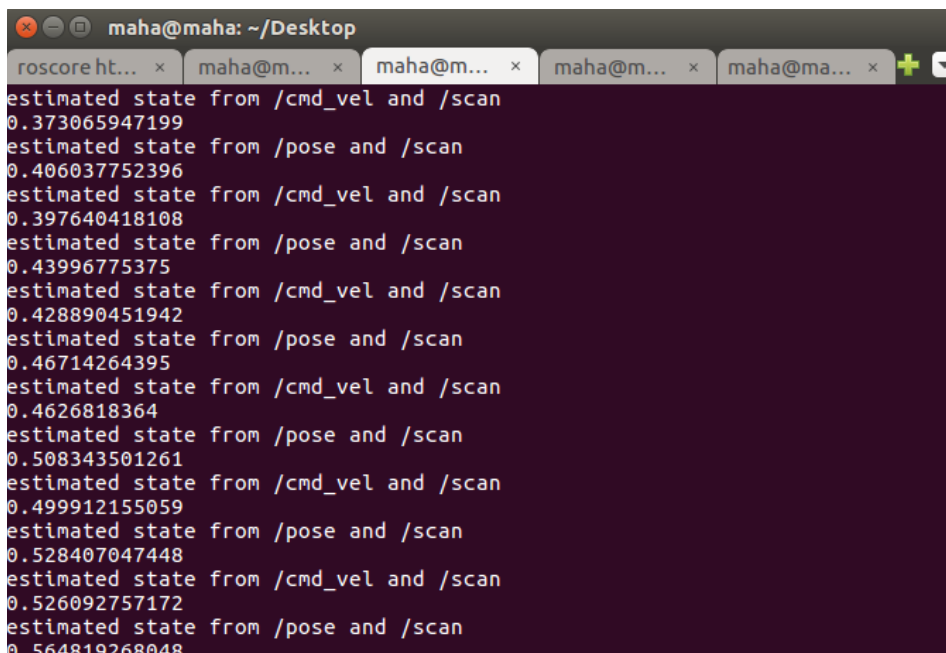
Finally, after calculating the estimated state, a topic called `/StateEstimatePose` has been created and used to publish the estimated state using `PoseWithCovarianceStamped` message. Below is a screen shot from the output of running: `rostopic echo /StateEstimatePose`.

A terminal window titled 'maha@maha: ~/Desktop' showing the output of 'rostopic echo /StateEstimatePose'. The output is a JSON-like structure for a PoseWithCovarianceStamped message. The first message has seq: 15, stamp: secs: 1580797350, nsecs: 461111068, frame_id: "odom2". The pose is at position (0.319342088983, 0.0, 0.0) with orientation (0.0, 0.0, 0.0, 0.0). The covariance matrix is a 6x6 identity matrix scaled by 2.6180339887543127. The second message has seq: 16.

```
---
header:
  seq: 15
  stamp:
    secs: 1580797350
    nsecs: 461111068
  frame_id: "odom2"
pose:
  pose:
    position:
      x: 0.319342088983
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0
  covariance: [2.6180339887543127, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
header:
  seq: 16
```

Illustration b: Publishing the estimated states

Also, the following is a screen shot from printing out the estimated state from both `/cmd_vel` and `/pose`:

A terminal window titled 'maha@maha: ~/Desktop' showing a sequence of state estimates. It alternates between 'estimated state from /cmd_vel and /scan' and 'estimated state from /pose and /scan'. The /cmd_vel estimates are linear velocities (x, y) and the /pose estimates are poses (x, y, theta).

```
estimated state from /cmd_vel and /scan
0.373065947199
estimated state from /pose and /scan
0.406037752396
estimated state from /cmd_vel and /scan
0.397640418108
estimated state from /pose and /scan
0.43996775375
estimated state from /cmd_vel and /scan
0.428890451942
estimated state from /pose and /scan
0.46714264395
estimated state from /cmd_vel and /scan
0.4626818364
estimated state from /pose and /scan
0.508343501261
estimated state from /cmd_vel and /scan
0.499912155059
estimated state from /pose and /scan
0.528407047448
estimated state from /cmd_vel and /scan
0.526092757172
estimated state from /pose and /scan
0.564819268048
```

Illustration c: Printing the estimated states

Evaluation: The state estimation has been implemented correctly. However, while the pose changes from 0 to 0.96 meter, the estimated state changes from 0 to 1.4 meter because of the uncertainty.

b) In task b, the transformation between (odom_kf frame) and (base_footprint frame) has been published to the /tf topic using a TFMessage message in which the parent_frame_id has been initialized to 'odom_kf' and the child_frame_id has been initialized to 'base_footprint'. Also the transform.translation.x variable has been set to estimated state. After publishing the message, the (run rqt_tf_tree rqt_tf_tree) command has been run to display the transformation tree (see the figure below).

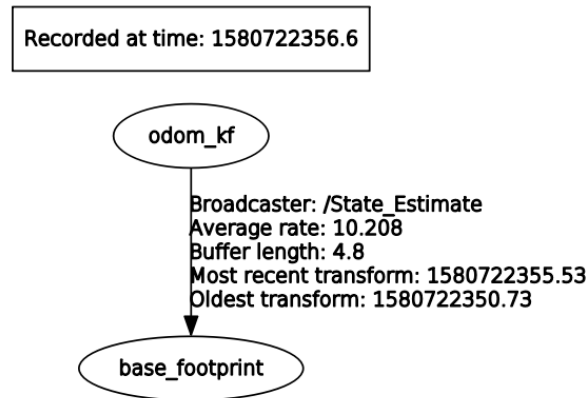


Illustration d: Transformation between frames

As can be seen from the above figure the parent frame is 'odom_kf' and the child frame is 'base_footprint'.

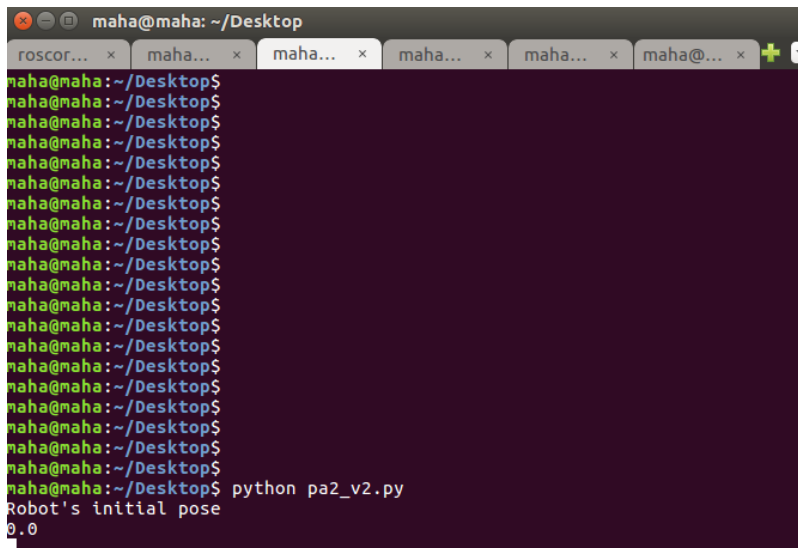
Also when I ran: (rostopic echo /tf) command, the output was as follows:

```
maha@maha: ~/Desktop
roscore ht... x maha@m... x maha@m... x maha@m... x maha@ma... x
---
transforms:
-
  header:
    seq: 0
    stamp:
      secs: 1580784411
      nsecs: 579027891
    frame_id: "odom_kf"
  child_frame_id: "base_footprint"
  transform:
    translation:
      x: 0.800520724947
      y: 0.0
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
---
transforms:
-
  header:
```

Illustration e: Publishing the transformation between odom_kf and base_footprint

Evaluation: I managed to publish the transformation between the 'odom_kf' frame and the 'base_footprint' frame successfully.

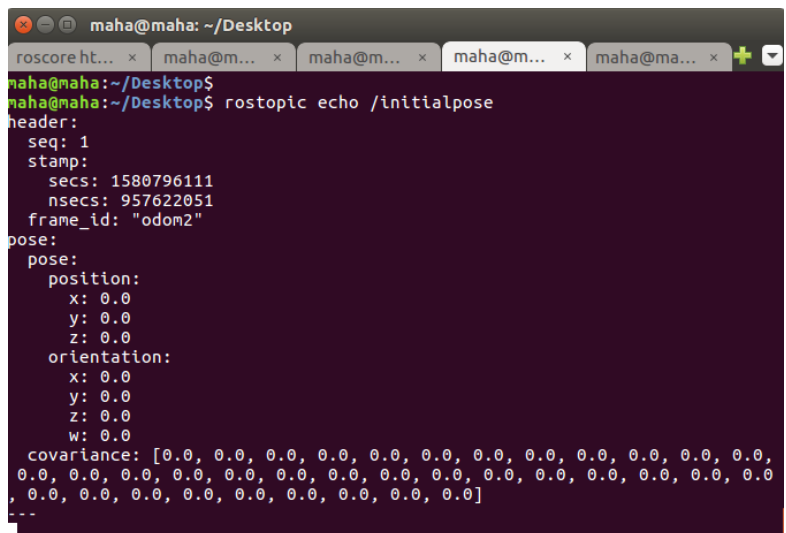
c) In task c, a subscriber has been also implemented to listen to the /initialpose topic so that the initial pose can be provided. An initial pose can be provided either using the command (rostopic pub /initialpose /PoseWithCovarianceStamped) or through rviz. As can be seen from the below picture, after publishing zero pose to the /initialpose topic, the output from running the implemented node which subscribes to /initialpose topic and prints the incoming message to the terminal is as follows:



```
maha@maha: ~/Desktop
roscor... x maha... x maha... x maha... x maha... x maha@... x
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$
maha@maha:~/Desktop$ python pa2_v2.py
Robot's initial pose
0.0
```

Illustration f: Printing the initial pose

Also, the following figure is the output of running (rostopic echo /initialpose) to see the full message not only the x pose.



```
maha@maha: ~/Desktop
roscor ht... x maha@m... x maha@m... x maha@m... x maha@ma... x
maha@maha:~/Desktop$
maha@maha:~/Desktop$ rostopic echo /initialpose
header:
  seq: 1
  stamp:
    secs: 1580796111
    nsecs: 957622051
  frame_id: "odom2"
pose:
  position:
    x: 0.0
    y: 0.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 0.0
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
```

Illustration g: The full initail pose message

Evaluation: I managed to successfully subscribe to the /initialpose topic and print the incoming pose to the terminal. I also created a publisher to the same topic for the sake of testing that the node can subscribe properly to the /initialpose topic and print the expected initial pose.

Question2:

Goal: Evaluate how good is your pose estimate. With the collected data in the first programming assignment, plot a graph with the following curves/points with different colors:

- The path estimated by the pose topic.
- The path estimated by the Kalman Filter, using cmd_vel and scan
- The path estimated by the Kalman Filter, using pose and scan
- (only for graduate students) The path estimated by the Kalman Filter, using cmd_vel and scan extracted by the RGB-D camera using the depthimage_to_laserscan
- (only for graduate students) The path estimated by the Kalman Filter, using pose and scan extracted by the RGB-D camera using the depthimage_to_laserscan
- The two measured points “as ground truth”.

Plot also the error (e.g., using matplotlib <https://matplotlib.org/>) between the initial point and end point of a., b., c., (d., e. only for graduate students) with respect to f.

Solution:

This Task has been done using the matplotlib library in python to plot the estimated states and the error in these estimations in different scenarios.

a, b, c,f) The figure below represents the plot of the estimated state using the pose only (shown as the blue colored line), the estimated state using both cmd_vel and scan (shown as the green colored line) and the estimated state using both pose and scan (shown as the red colored line). Also the error between the initial point of the measured point (ground truth) and the initial points in the previous three lines (a,b,c) as well as the error between the end point of the measured point (ground truth) and the end points in the previous three lines (a,b,c) can be seen as 6 points in the below figure. The black circles represent the ground truth itself, the blue x represent the error between the ground truth and the initial and end points of the blue line, the red triangles show the error between the ground truth and the initial and end points of the red line and finally the green pluses represent the error between the ground truth and the initial and end points of the green line.

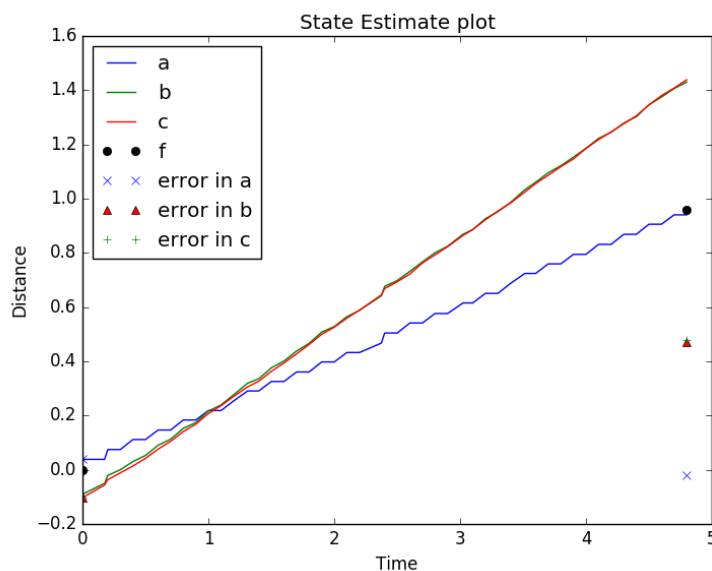
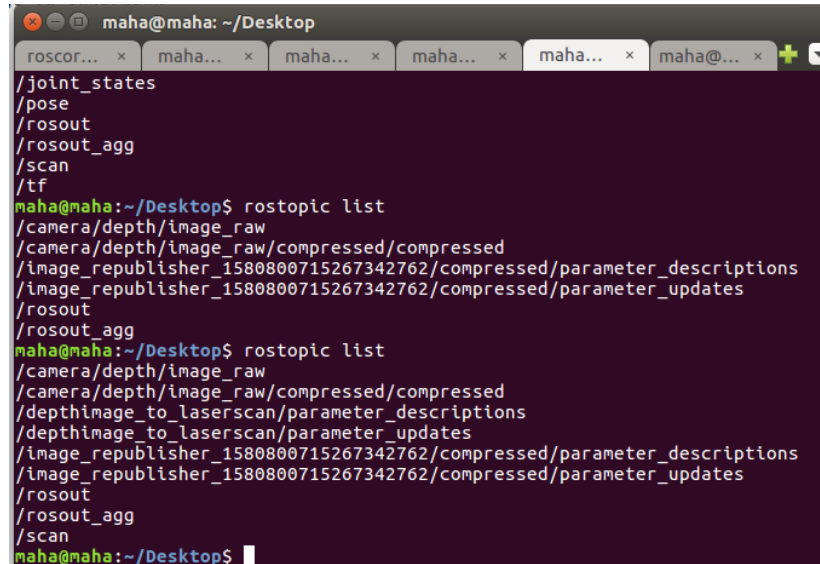


Illustration h: plotting state estimation using the LIDAR

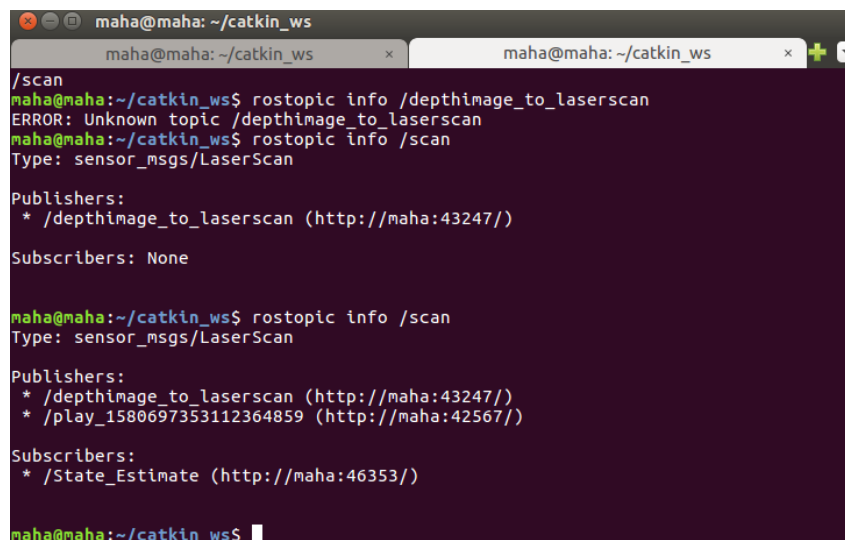
d,e,f) In these final tasks, I needed to first decompress the camera depth images using the following command (roslaunch image_transport republish compressed in:=/camera/depth/image_raw/compressed raw out:=/camera/depth/image_raw/). The following figure shows the output from running (rostopic list) after the decompression. As can be seen /camera/depth/image_raw is now one of the provided topics.



```
maha@maha: ~/Desktop
roscor... x maha... x maha... x maha... x maha... x maha@... x
/camera/depth/image_raw
/pose
/rosout
/rosout_agg
/scan
/tf
maha@maha:~/Desktop$ rostopic list
/camera/depth/image_raw
/camera/depth/image_raw/compressed/compressed
/image_republisher_1580800715267342762/compressed/parameter_descriptions
/image_republisher_1580800715267342762/compressed/parameter_updates
/rosout
/rosout_agg
maha@maha:~/Desktop$ rostopic list
/camera/depth/image_raw
/camera/depth/image_raw/compressed/compressed
/depthimage_to_laserscan/parameter_descriptions
/depthimage_to_laserscan/parameter_updates
/image_republisher_1580800715267342762/compressed/parameter_descriptions
/image_republisher_1580800715267342762/compressed/parameter_updates
/rosout
/rosout_agg
/scan
maha@maha:~/Desktop$
```

Illustration i: Decompressing the camera depth images

Then, I ran the following command (roslaunch depthimage_to_laserscan depthimage_to_laserscan) to transform the depth image into a scan information. The below figure represents the output of running (rostopic info /scan) which shows the depthimage_to_laserscan as the publisher of the LaserScan message in my node.



```
maha@maha: ~/catkin_ws
maha@maha:~/catkin_ws x maha@maha:~/catkin_ws x
/scan
maha@maha:~/catkin_ws$ rostopic info /depthimage_to_laserscan
ERROR: Unknown topic /depthimage_to_laserscan
maha@maha:~/catkin_ws$ rostopic info /scan
Type: sensor_msgs/LaserScan

Publishers:
* /depthimage_to_laserscan (http://maha:43247/)

Subscribers: None

maha@maha:~/catkin_ws$ rostopic info /scan
Type: sensor_msgs/LaserScan

Publishers:
* /depthimage_to_laserscan (http://maha:43247/)
* /play_1580697353112364859 (http://maha:42567/)

Subscribers:
* /State_Estimate (http://maha:46353/)

maha@maha:~/catkin_ws$
```

Illustration j: Transform from depth image to laser scan

Finally, I plotted all the figures again but this time using distances from the camera depth instead of the LIDAR. As can be seen from the figure below, one can parley find a difference between the figures

from the LIDAR and the figures from the camera depth. From this we can conclude that the depth from the camera is very accurate and can provide us with almost the same measurements as the LIDAR.

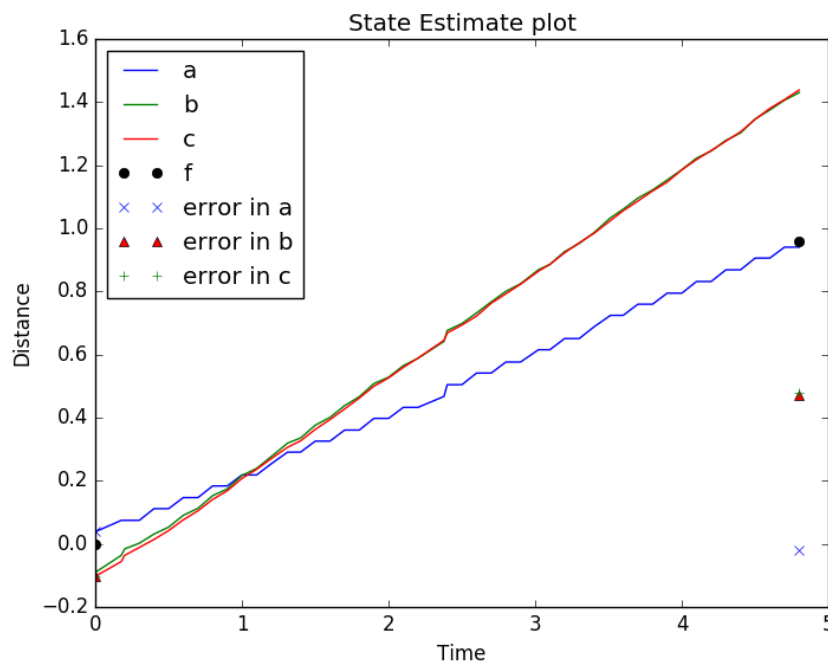


Illustration k: plotting state estimatio using camera depth

Evaluation: I managed to plot all the required figures properly and to show the accuracy by which the camera depth can provide distances.

Allocation of effort:
I did everything alone.