**Automated Error Diagnosis Tool Clafer Compiler**

ECE750-T24

Matthew Ma, Jesus Jesus Alejandro Padilla

Department of Electrical and Computer Engineering, School of Computer Science

University of Waterloo

Waterloo, Ontario

*Abstract*— Software bugs have been causing tremendous .

Keywords-component; state estimation; motion model; measurement model; extended Kalman filter; mapping; occupancy grid mapping; inverse measurement model; log odds raito; LIDAR; four-wheeled ground robot

# Introduction

From the work started in “Motion Control of a Four-Wheeled Ground Robot” [1], the motion was defined and completely controlled for the Clearpath Robotics© Chameleon R100. System identification provided a complete overview of the physical dynamics of the robot, while the velocity and heading controllers implemented provided complete control over the position and orientation of the robot.

However, in order to facilitate more advanced concepts and functionalities, such as mapping, it was necessary for the robot to be able to obtain a sense of its true state. To achieve this objective, the position and orientation of the robot within a set area was obtained using the indoor positioning system (IPS). This data, in conjunction with data obtained from wheel encoders and the previously defined motion models, was used to obtain an estimate of the state. However, upon analyzing the results it appeared the state estimates were prone to noise that would potentially cause issues later on. To mitigate this potential problem, an Extended Kalman Filter (EKF) was implemented.

The implementation of the EKF, along with the velocity and heading controllers developed, would allow control of the position of the robot and the ability to accurately predict the state of the robot with the measurement data. The surroundings were determined by using a Hokuyo Light Detection and Ranging (LIDAR) module mounted at the front of the robot. The LIDAR sweeps the environment using laser and collects the range of detected object by measuring the time of flight for the sent laser to return. Then an inverse measurement model was used to combine the scan data into an occupancy grid map with a predefined size. Log odds ratio and Bayesian filter update rule was incorporated to generate a final belief map.

# Motion & Measurement Models

To gain a true understanding of the motion of the robot, a state model was defined that incorporated the physical dynamics of the robot as well as the sensor measurements.

To define the motion of the robot, the simple bicycle model was used as the basis for development [2]. However, this model was now augmented to include a state to represent the velocity of robot in the global coordinate frame. This additional state was required, as the inputs to the robot are a commanded motor throttle and steering angle.

The update equation for this additional state was also defined using the motor transfer functions defined in “Motion Control of a Four-Wheeled Ground Robot” seen in (1) below.

(1)

The z-domain equivalent was then found by using a Tustin approximation given a 20 Hz sampling rate, seen in (2) below.

(2)

This equation was then transformed back to the time domain and the constants were gathered to form a signal constant, Km. This constant incorporates the motor torque constant, motor time constant and sampling time into a single value. The resulting update equation is seen in (3) below.

(3)

Given (3) and the bicycle model presented in [2], a completed motion model for the system could be defined in (4) and (5) below where represents Gaussian noise.

(4)

(5)

With the motion model defined, a measurement model was developed to correlate the data obtained from the sensors to the robot state. Given that the indoor positioning system (IPS) returns data directly as x and y coordinates, states 1 and 2 were directly fed in. However, state 3 is returned in units of degrees and thus needed to be converted into radians. Lastly, state 4 was returned as encoder ticks and was converted to velocity through means of a constant . The velocity would then be multiplied by 0.1, as this was the encoder update rate, to obtain the number of ticks. Thus, the measurement model was defined in (6) below where represents Gaussian noise.

(6)

Thus the non-linear motion model and linear measurement model were defined. These definitions proved invaluable for the implementation of the Extended Kalman Filter.

# Extended Kalman Filter

To gain further confidence in the sensor data an Extended Kalman Filter (EKF) was implemented to provide an accurate estimate of the true state of the robot. This was done to filter any noise that would have caused erroneous data to be propagated to the controller.

## Extended Kalman Filter Development

The extended Kalman filter is derived off of the framework of the Kalman Filter [3]. However, for the EKF some of the requirements for the Kalman filter are relaxed to allow application in non-linear systems. This relaxation of the requirements comes at the cost of optimality in the estimate generated by the Kalman Filter.

To implement the EKF algorithm, the partial derivatives of the motion and measurement models, with respect to each of the states, needed to be taken to form the G and H matrices respectively. By taking the partial derivatives of the system in this manner the motion and measurement models were linearized about a point. Furthermore, the 4\*1 motion model matrix was converted to a 4\*4 matrix, allowing many of the calculations required for the EKF to be completed. The resulting partial derivatives can be seen in (6) – (9) below, where *dt* is the system update rate and lrobot is the length of the back axle of the robot

.

(6)

(7)

(8)

(9)

As the measurement model is linear to begin with, the linearization of the measurement model did not change the size of the resulting matrix. Thus, the linearized measurement matrix produced can be seen in (10).

(10)

Once the linearized matrices were obtained, associated covariances for the robot sensors were required. These covariances were obtained by placing the robot in various locations for periods of time while the position and heading data was collected. The velocity covariance was obtained by removing the velocity controller and setting the robot to 100% throttle for fixed periods of time. This test was completed several times to collect various data sets. This data was then analyzed offline to obtain estimates for the covariance of each of the state measurements. These results are summarized in Table 1 below.

1. Covariance Values

| Covariance Values | | | |
| --- | --- | --- | --- |
| X Position(m) | Y Position(m) | Heading(rad) | Velocity(m/s) |
| 3.69\*10-10 | 2.68\*10-10 | 8.53\*10-8 | 0.00156 |

Once the covariances were obtained, matrices for the motion model noise and measurement noise, R and Q respectively, were defined to be implemented in the EKF algorithm. The predicated state, in the absence of noise, was then calculated using the motion model defined above and the last estimate of the state. The EKF algorithm described in [4] was then implemented using the equations outlined in (11) – (15).

(11)

(12)

(13)

(14)

where (15)

These equations use the predicted state, mean and covariance to calculate a Kalman gain Kt. This gain is then used to produce an estimate of the current state of the robot which can then be propagated forward to the system controllers.

The simplifications of the EKF are only feasible because of the Markov Assumptions [5] that claim that the state of the system can be obtained using only the information from the previous time step, or in this case, its estimate and the inputs of the current time step. These simplifications, along with the linearization, allow for a reasonable estimate of the state to be obtained.

## Extended Kalman Filter Simulation

To validate the EKF developed above a simulation was developed using the Ackerman Steering models developed in “Motion Control of a Four-Wheeled Ground Robot” and the motion and measurement models presented above. The values used in the simulation mirrored those of the physical robot and thus allowed for an accurate test of the EKF’s ability to filter unwanted noise. The simulation assumes that the robot is running at 100% torque in a straight line with no corrections. The output of the simulation, seen in Fig. 1 below, showed the estimated and true state of the robot as it moved in space.

C:\Users\Eugene's Laptop\Dropbox\ME 597 Labs\Lab 2\EKF\EKFSim.tif

1. EKF Simulation

However, to truly test the capabilities of the EKF large measurement noise, on the order of 10-1, was assumed to exist in the system. This came from the belief that the motion model developed above was fairly accurate whereas the values returned by the sensors was not; the results of these tests can be seen in Fig.2.

C:\Users\Eugene's Laptop\Dropbox\ME 597 Labs\Lab 2\EKFTest.tif

1. EKF Simulation Results

As the figure shows, the estimate, shown by the solid line, was able to provide an accurate estimate of the state of the robot in the face of very noisy sensor data, shown by the thin lines. This indicates that the algorithms behind the EKF are able to deal with incorrect sensor data if it does exist.

## Experimental Implementation

The algorithms developed and tested in simulation were modified slightly when being implemented in the physical experiment. In the simulation environment the entire state, all measured values and all matrices required for the EKF were stored in memory from the beginning of the simulation until the end. This was possible as the simulation was run for fixed periods of time and memory was not an issue. However, as the experimental configurations were not designed to run for set periods of time, these algorithms were modified slightly to optimize the memory required. Rather than store all information from initialization, only the information required for the current estimate and previous estimate were kept.

Only the current and previous information was kept due to the fact that only this information was required to predict the state of the robot. This is, once again, due to the simplifications made in the Markov Assumptions [5].

## Experimental Data Evaluation

To validate the EKF implemented on the robot, a similar test to those implemented in simulation, 100% throttle and no zero, was run on the robot with the true covariances of the sensors used. However, due to the high accuracy of the sensors, indicated by the low covariances, the measured and estimated states do not vary greatly; this is seen in Fig. 3 below.

C:\Users\Eugene's Laptop\Dropbox\ME 597 Labs\Lab 2\SimEKF.tif

1. EKF Experimental Results

The low covariance in the sensor data indicates that the noise disturbances seen in the vehicle motion are not attributed to sensor noise but to non-linearities in the physical system that force the robot to deviate from its predicated path. These non-linearities, dirt on the floors for example, cannot be modeled accurately to any degree and thus cannot be effectively eliminated. Although there does not seem to be any degraded performance from implementation of the EKF, the ability of the EKF to reject noise seen in simulation is not wholly apparent in the experimental data. However, if less efficient and accurate sensors were used the benefits observed from the EKF are likely to be substantial.

# Mapping

## Mapping Developments

The LIDAR measurement was accessed through the Hokuyo package on ROS. At each scan, a vector of 512 float elements was returned with the numerical value representing the range information in meters. The angular scan range in this particular Hokuyo LIDAR is π radians and has a maximum range of 4 meters.

To properly relate the data returned by LIDAR, the inverse measurement model was used to localize the scan data by fusing the measurement with the current position estimate of the robot. An occupancy grid map was then used to store the environment scans.

The inverse measurement successively iterates through each of the occupancy grids and calculates the range and bearing with respect to the robot state [6], where is the x position, is the y position, and is the yaw at time t.

(16)

(17)

Based on the range and bearing calculated for each grid cell, the associated LIDAR measurement in range are selected if available, and the following probability values are assigned,

* object is not likely be present – 0.3
* no new information is obtained – 0.5
* an object is likely to be present – 0.7

The log odds ratio of the probability is then applied to the entire grid map, shown as the logit (logistic regression) function,

(18)

This log odds ratio result represents the probability of an object existing at a particular position by converting it to a log scale. After the log odds ratio is applied, the Bayesian Filter update rule is applied to aggregate the current measurement belief with the previous belief map, as described by (19),

(19)

Using Markov’s assumption, the Bayesian log odds update rule can be rewritten as (20) [7],

(20)

In short, the map update rule can be written as (21),

(21)

, where

: Previous log odds belief map

: Initial belief map

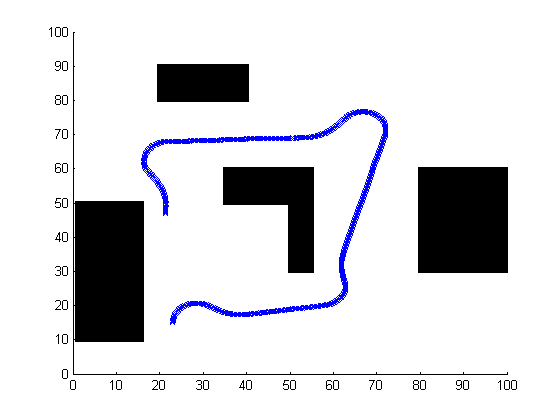
: Current map as constructed by inverse measurement model

The belief map is a grid that provides a description of the probability of an object’s presence at a particular coordinate in the grid. As the robot traverses through the surroundings and collects more scans with the LIDAR, it collects more data and improves the confidence in the probability of the surroundings. The areas that the algorithm determines to contain objects will eventually reach values approaching or equal to one, while areas that are deemed to be empty will approach 0.

## Mapping Simulation

The algorithm was implemented in Matlab simulation to fully test the system. An artificial map was generated with obstacles placed at various locations of the map for detection purposes. Four way points were specified for the robot to traverse. Since obstacle avoidance was not the focus of this exercise, the obstacles and robot trajectory were designed to not interfere for demonstration of the results from the inverse measurement and mapping algorithm.

The map size was determined to be 100x100 grid size and the same would be implemented in the physical robot setup. The maximum range of the LIDAR scan was arbitrarily selected to be 20 units. Fig.4 shows the trajectory of the robot reaching each of the four predefined way points.



1. Robot Trajctory in Simulated Environment

To appropriately simulate the LIDAR measurement, a function based on the ray-tracing algorithm was used to return the simulated measurement array needed to test the inverse measurement model [8].



1. Updated Belief Map

Fig. 5 shows the final result of the belief map, combining all the results of the inverse measurement model obtained at each time step. The result was satisfactory, where the outlines of the obstacles were accurately detected and mapped.

## Mapping Implementation

The algorithm developed and tested on simulation was modified slightly to cope with the real time processing constraints on the Chameleon R100 robot. Similar to simulation, an occupancy grid matrix of size 100x100 was updated at each time step update.

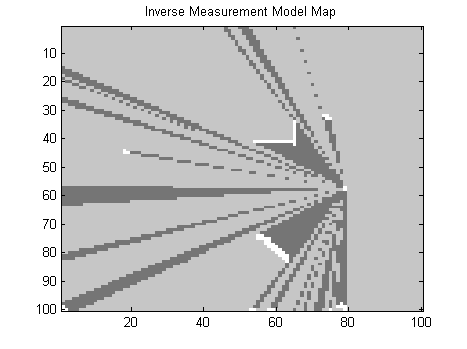
In simulation, the environment that the robot operates in was assumed to be the same scale as the map size. Due to the physical limitation of the IPS, the range of trackable area was smaller, approximately a square measuring 3.5 meters by 3.5 meters. Also, unlike simulation, the origin of the tracked square from the IPS was not at the lower left corner. Therefore, a scaling factor and an offset were applied to the proposed algorithm to convert the map’s grid index to physical location from the IPS. The measurement model was adjusted appropriately to calculate this conversion.

Since the steering controller, velocity controller, and EKF were implemented on a single thread and were updated at a rate of 10 Hz, the integration of the mapping algorithm in the same thread was deemed non-ideal as the computational requirement was expected to grow rapidly as the map size was increased. To allow maximum flexibility in both the size and update rate in mapping, a separate thread specifically for mapping was implemented.

## Experimental Data Evaluation

The final implementation used a grid of 100 x 100 to map an area of size 3.5 meter x 3.5 meters. The robot was commanded to move straight at 30% throttle for a total of 30 seconds. The implemented mapping thread was updated at a slightly reduced rate of 4 Hz to ease the amount of processing power consumed. To facilitate testing and debugging, the belief map, as well as several intermediate calculation matrices, were saved to .csv files for offline viewing and analysis.

Unlike the calculated LIDAR measurement obtained in simulation, the actual measurements obtained from the robot contained a fair amount of noise. It was found that zero was returned if the laser information never returned to the LIDAR. These values were ignored in the inverse measurement model thus leaving sections of angular ranges of empty measurements. Fig.6 illustrates the gaps in measurement rays.

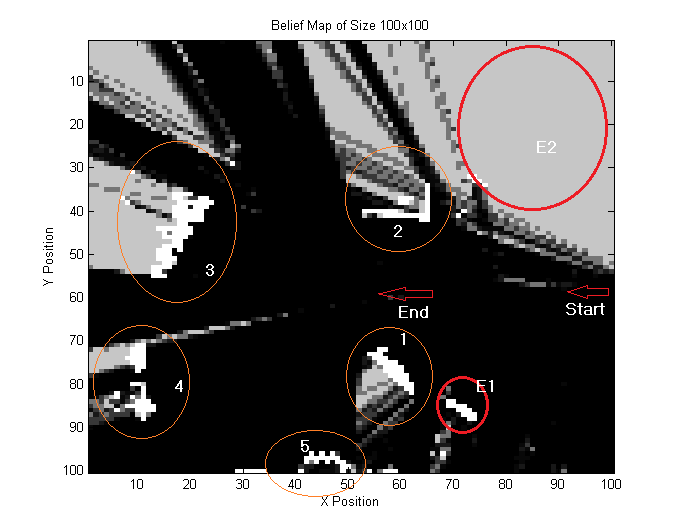


1. Inverse Measurement Model Map at Single Time Step

Finally, combining all the inverse measurement model results and applying log odds to update final map. Fig. 7 illustrates the result of the mapping algorithm. The red arrows represent the starting and ending position of the robot. The features highlighted in yellow are features detected as it can be compared with Fig. 8 The features identified can be summarized as,

1. Grey carry-on box placed at 45 degree angle
2. Green box
3. Husky A100 robot placed at -30 degree angle
4. Swivel Chair (not shown in Fig.8)

From analyzing the belief map, two anomalies were detected: a large empty spot at the top right corner (E2) and the mysterious object next to the carry-on box (E1), both highlighted in red. Careful investigation of the system log files shows the yaw information provided by the IPS is erroneous in the first fourteen seconds of the test, as the yaw measurement should be approximately +90 degrees. This is seen in Fig. 9.



1. Belief Map



1. Actual Test Setup
2. Measured Yaw vs. Time

As a result, the anomalies seen in E1 and E2 can be accounted to the wrong yaw information provided by the IPS. This explains the large blank spot at E2 in which the LIDAR measurements are all erroneously associated to the bottom half of the map. The location of E1 can also be adjusted by correcting the measured yaw of the robot and re-computing the inverse measurement model and log odds map update.

# Conclusions

Given the ability to accurately model the motion of the robot and additional confidence from the highly accurate sensors, an estimate of the state is capable at any time step, given the previous state information. As demonstrated by the simulation results, if either of these resources were to fail individually, the implemented EKF could still provide a fairly accurate estimate of the vehicle state for further use.

However, given the accuracy of the sensors, the implemented EKF shows little improvement to the general performance of the robot. The filter however, does not degrade the performance and is left in place in the event of unforeseen issues.

Furthermore, a belief map based on the inverse measurement model and occupancy grid was successfully constructed using 2D LIDAR measurement and robot state tracking information provided by IPS. Although the experimental map does not completely resemble the environmental setup due to variation in IPS data, the test result demonstrates the accurate execution of the mapping algorithm using inverse measurement model and log odds function. The test result also shows the disadvantage in this mapping method as the integrity of the occupancy map is highly dependent and sensitive to the accurate measurement of both the position and orientation of the robot.

##### References

1. Hui. M, Liang. R, Li. E, Wong. T., “Motion Control of a Four-Wheeled Ground Robot,” ME597, University of Waterloo, September 2011.
2. Waslander S., “ME 597: Motion Modelling”, ME 597 Lecture Slides, University of Waterloo, pg. 65, October 1st 2010
3. Waslander S., “ME 597: Estimation”, ME 597 Lecture Slides, University of Waterloo, pg. 100, October 1st 2010
4. Waslander S., “ME 597: Estimation”, ME 597 Lecture Slides, University of Waterloo, pg. 106, October 1st 2010
5. Waslander S., “ME 597: Motion Modelling”, ME 597 Lecture Slides, University of Waterloo, pg. 29, October 1st 2010
6. Waslander S., “ME 597: Mapping I”, ME 597 Lecture Slides, University of Waterloo, pg. 38, October 1st 2010
7. Waslander S., “ME 597: Mapping I”, ME 597 Lecture Slides, University of Waterloo, pg. 30, October 1st 2010
8. Waslander S., “ME 597: Mapping I”, ME 597 Lecture Slides, University of Waterloo, pg. 42, October 1st 2010

This page intentionally left blank