Simulataneous Robot Localization and Environment Mapping with Simulated Sensor Measurements

Swapnil Pande and Joshua Petrin, *EECE 3892-04, Dr. Richard Alan Peters*

***Abstract*—The abstract goes here.**

I. Introduction

FOR mobile robots, self-localization is extremely important. A robot must know its current state to properly perform its duties and maintain its operation. However, localization is non-trivial due to inherent inaccuracies in actuators and sensor systems, as well as non-ideal environments, which introduce additional errors. Robotic mapping of unknown environments is equally important and poses similar problems as localization. Given that the robot location is known, it is difficult to build an accurate map of the environment due to noisy sensor measurements.

Mapping and localization are complementary problems. Localization depends on having an accurate map of the environment to which to compare sensor measurements. Mapping relies on exactly knowing the robot location in order to accurately place features on a map. However, in most robotic applications, neither the precise location of a robot nor a map of the environment are known.

In order to address this problem, Simulataneous Localization and Mapping (SLAM) techniques have been developed, which rely on probabilistic methods to build probability distributions of expected positions of features and the robot in a map. SLAM algorithms have a vast number of potential applications. For instance, they have been used to develop drone and land-based robotic systems capable of mapping disaster zones [1], [2]. They are being used in autonomous cars to safely navigate traffic. Also, they have been used to build volumetric maps of complex mines [3].

There are three main paradigms for solving the SLAM problem: The Kalman Filter, the Particle Filter, and Graph-based solutions. The Kalman Filter is computationally the least intensive of these techniques, as it parameterizes the distributions, assuming that the robot pose and landmark positions follow a Gaussian distribution. The Particle Filter utilizes Monte Carlo techniques to estimate the state of the robot. Graph-based solutions assume the objects around the robot can be approximated to a course grid landscape. [4]

This project explores the Kalman Filter and its extension, the Extended Kalman Filter (EKF), for running a SLAM simulation. The authors believe this is best for practical application because most robots can implement the EKF with relative computational ease.

The objective of this project was to apply SLAM algorithms to build autonomous capabilities for a robot designed to compete in the NASA Robotic Mining Competition. The robot is required to mine a subsurface icy-simulant (gravel) in a Martian-like environment and deposit it into a collector bin. The robots performance is scored based on the amount of simulant collected within a 10 minute period in addition to performance criteria such as efficiency mass, energy consumption, communication bandwidth usage, and dust kickup during the mining operation.

The competition field is a 7.38m x 3.88 m rectangle as shown in Figure 1. The robot begins the match in the start zone with an unknown position and orientation and must traverse the obstacle area to the mining area to collect the gravel. The gravel must be returned to the collector located adjacent to be counted towards the score of the match. The obstacle area will contain three obstacles, randomly placed in the zone. The diameter of each obstacle may range from 10-30 cm and the mass may range from 3-10 kg. A target or beacon may be placed on the collector bin as a landmark for localizing the robot.

The robot is driven by 4 wheels, each with an independent drive motor and encoder. Due to design constraints, the robot is not capable of traversing the obstacles. The robot will be equipped with the following sensors:

- A Microsoft Kinect to map the environment
- A camera with software capable of determining the location of a fiducial marker
- An Inertial Measurement Unit to estimate pose of the robot

The autonomy algorithm must be capable of the following tasks:

1) Determining the initial pose of the robot based on data collected from the environment
2) Traversing the obstacle zone while avoiding obstacles.
3) Determining when the robot has reached the mining zone and collecting gravel.
4) Returning to the start area and depositing the gravel in the collector bin
5) Repeating the mining process multiple times to maximize the gravel collected while avoiding holes created from other mining runs.

For the purpose of this project, the scope of the challenge was limited to localizing the robot and determine the location of obstacles to avoid. It is assumed that a path planning

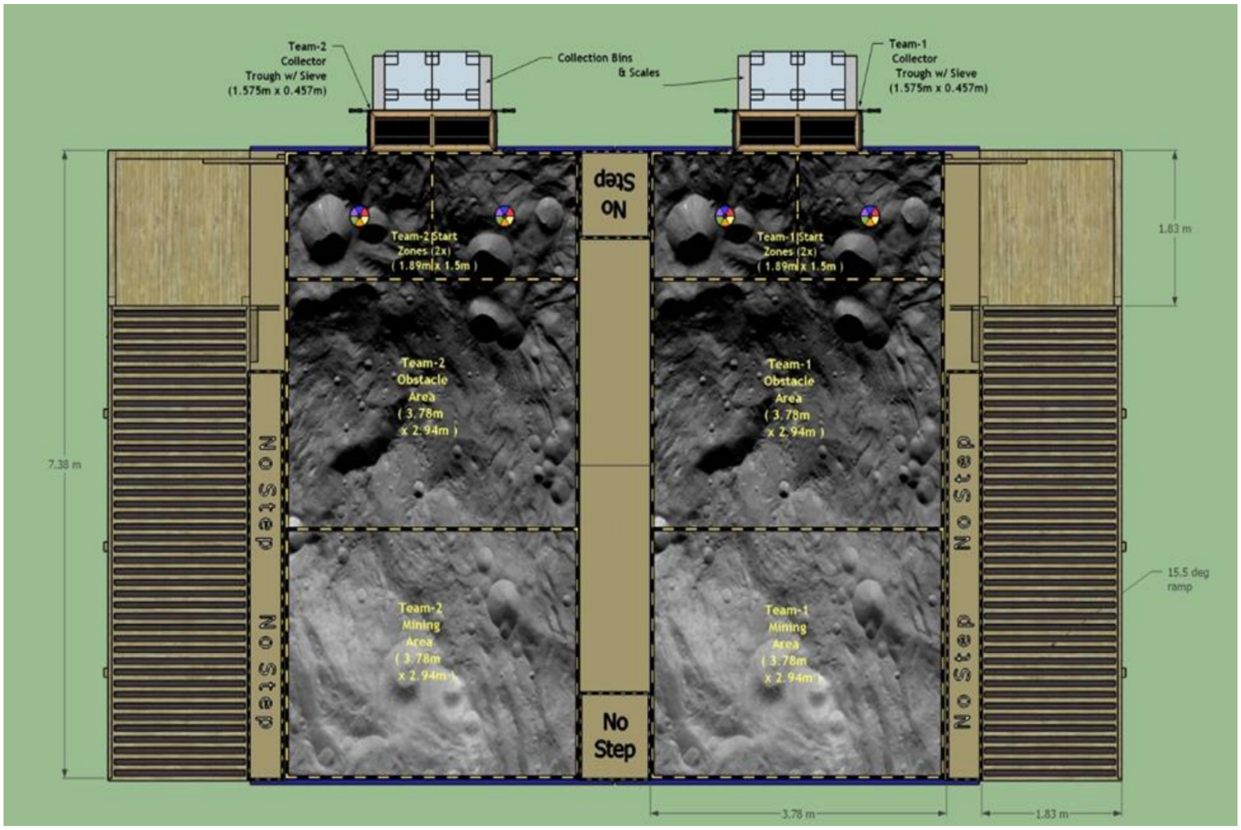

Fig. 1. Diagram of competition field [5]

algorithm will be capable of providing control inputs to optimize the path of the robot. This objective is achievable through the application of a SLAM algorithm.

II. Methods and Procedures

The simulation for the EKF SLAM algorithm is demonstrated through an abstract robot which can sense the two-dimensional environment through multiple types of sensors. To simulate the NASA RMC robot, the robot can sense the state of itself and the environment obstacles through multiple abstracted types of sensors, including:

- A Microsoft Kinect, which returns a range-bearing measurment to all obstacles.
- A gyroscope, which can determine the robots angular orientation relative to its previous frame.
- A camera capable of determining the position of a fiducial marker placed on the collector bin.
- Encoders, which can determine the robots location relative to the previous frame.

The Robot class (in Robot.h) represents the actual position of the robot inside its environment. From here, the robot can make “measurements” with respect to its various sensors. These measurements, which can be called from methods such as `getGyroMeasurement()` and `getArucoMeasurement()`, simply return the respective quantities the sensor is trying to measure, plus zero-mean gaussian noise with a preset standard deviation. For instance, when the robot calls to its function `getArucoMeasurement()` while located at `(1, 1)`, the function may return the vector `(1.03, 0.99)` as a result of the applied sensor noise. The standard deviations were determined by collecting measurement samples from the various sensors and generating the sampling distribution.

Once these measurements are collected, the `EKFSlammer` class is instantiated. This class represents the robot’s predicted measurements by the Extended Kalman Filter. It also represents the EKF’s entire algorithmic complexity. The *a priori* and the *a posteriori* estimation steps are performed through several of the `EKFSlammer`’s class methods.

`EKFSlammer` stores the robot’s and environment’s state x_t in a `VectorXd`. The perceived position and orientation of the robot, as well as the location of the obstacles in the map, can be expressed informally as

$$x_t = \begin{bmatrix} \vec{s} \\ \theta \\ \vec{m}_1 x \\ \vec{m}_1 y \\ \vdots \end{bmatrix},$$

where $\vec{s}$ is the robot’s location with respect to the Aruco Marker (the arbitrarily-defined origin) and $\vec{m}_i$ is the location of the i^{th} obstacle. The vectors represented in code for measurement updates are the Kinect measurement vector and the gyroscope measurement vector. The Kinect senses all obstacles in the environment; therefore, its measurement vector for all

i obstacles is expressed as

$$z_{t,K} = \begin{bmatrix} r_1 \\ \theta_1 \\ r_2 \\ \theta_2 \\ \vdots \\ r_i \\ \theta_i \end{bmatrix} + \vec{\epsilon_t},$$

and the gyroscope measurement vector can be expressed as

$$z_{t,g} = \omega_t + \epsilon_t,$$

where ω_t is the angular velocity of the robot, and $\vec{\epsilon_t}$ and ϵ_t are the normally-distributed noise factors for each sensor.

To step through a frame with a set timestep, the `ekfUpdate()` function is called. Here, the main method passes its control input, the perceived obstacles (from the Kinect observation), the measured acceleration, the measured encoder values, the measured gyroscope values, and the measured Aruco Marker location. `ekfUpdate` then calls all other relevant EKF functions.

First, to generate the *a priori* estimate, the `motionModelUpdate()` method is called by `ekfUpdate`. Its parameters are the timestep for the robot and its control parameters, in the form of a forward velocity and an angular velocity (i.e. the `controlIn` struct; see `Utils.h`). The `motionModelUpdate` will recalculate the estimated state of the robot, given the control input and given the known uncertainties of the robot's actuators (i.e. the robot's wheels, since this robot can only move).

The motion model update for the *a priori* estimate is defined with the following relation, given the robot's control sequence straight-line velocity v_t and angular velocity ω_t:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} -v_t/\omega_t \sin(\theta_t) + v_t/\omega_t \sin(\theta_t + \omega_t \Delta t) \\ v_t/\omega_t \cos(\theta_t) - v_t/\omega_t \cos(\theta_t + \omega_t \Delta t) \\ \omega_t \Delta t \end{bmatrix}$$

If covariance of the current state is Σ_t, then the Extended Kalman Filter approximates the covariance of the new *a priori* state $\widehat{\Sigma_t}$ to be

$$\widehat{\Sigma_t} = G_t \Sigma_t G_t^T + R_t,$$

where G_t is the Jacobian of the robot's motion, and R_t is a noise factor [4]. Formally,

$$G_t = \begin{bmatrix} G_t^x & \mathbf{0} \\ \mathbf{0} & I_{2N \times 2N} \end{bmatrix},$$

where N is the number of obstacles (the quantity $2N$ above is because there are two coordinates for each obstacle) and

$$G_t^x = \begin{bmatrix} 1 & 0 & -v_t/\omega_t \cos(\theta_t) + v_t/\omega_t \sin(\theta_t + \omega_t \Delta t) \\ 0 & 1 & -v_t/\omega_t \sin(\theta_t) + v_t/\omega_t \sin(\theta_t + \omega_t \Delta t) \\ 0 & 0 & 1 \end{bmatrix}.$$

Second, to generate the *a posteriori* estimate, the `ekfUpdate` function calls the `ekfCorrectionStep`. The `ekfCorrectionStep` in turn integrates all of the sensor data it is passed to create a new model for the robot's and the obstacles' states. It does this by stepping through the Extended Kalman Filter algorithm's correction step with each set of

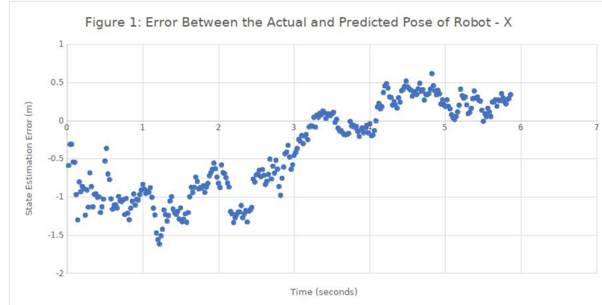

Fig. 2.

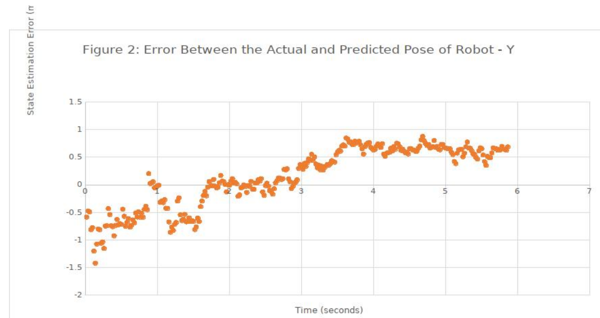

Fig. 3.

sensor data. For example, `ekfCorrectionStep` will call `kinectUpdate` and pass the data from the Kinect sensor to correct the positions of the obstacles.

The measurement update must calculate certain values to integrate the sensor data into the estimated state, such as the Kalman gain K_t:

$$K_t = \widehat{\Sigma_t} H_t^T (H_t \widehat{\Sigma_t} H_t^T + Q_t)^{-1},$$

where H_t is the Jacobian of each predicted sensor measurement.

The Kalman gain is a weight for the updated state and covariance to depend on the predicted state verses the observed state. The updated state is

$$x_{t+1} = \hat{x}_t + K_t (z_t - h(\hat{x}_t)),$$

and the updated covariance is

$$\Sigma_{t+1} = (I - K_t H_t) \widehat{\Sigma_t}.$$

This *a posteriori* correction step is performed for each set of sensor data to get an accurate value for the mean and covariance of the new state. [4]

III. Experimental Results

IV. Discussion

V. Conclusion

The conclusion goes here.

Appendix A

Appendix two text goes here.

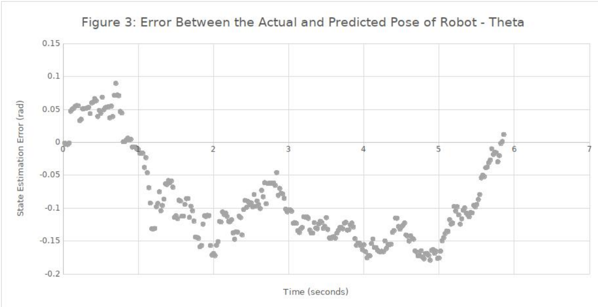

Fig. 4.

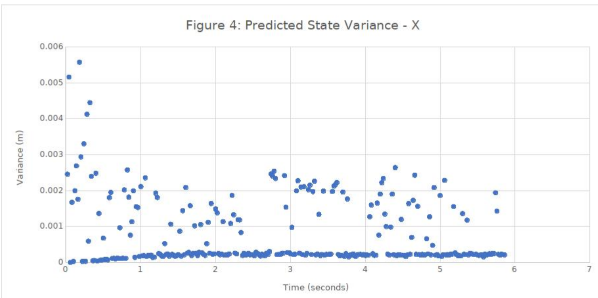

Fig. 5.

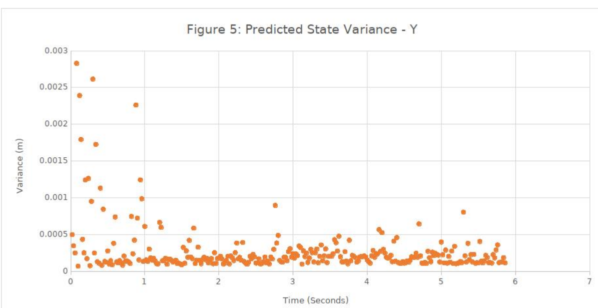

Fig. 6.

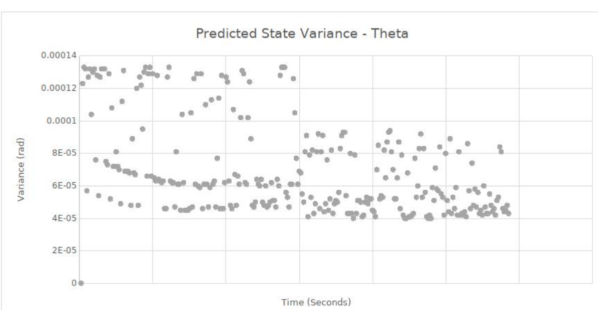

Fig. 7.

References

[1] C. Alex and A. Vijaychandra, “Autonomous cloud based drone system for disaster response and mitigation,” *2016 International Conference on Robotics and Automation for Humanitarian Applications (RAHA)*, Kollam, 2016, pp. 1-4.

[2] A. Kleiner, C. Dornhege and S. Dali, “Mapping disaster areas jointly: RFID-Coordinated SLAM by Hurnans and Robots,” *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*, Rome, 2007, pp. 1-6.

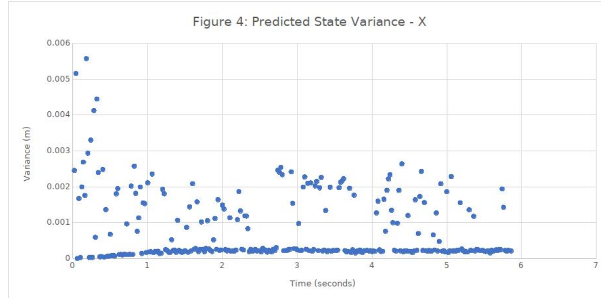

Fig. 8.

[3] A. Nuchter, H. Surmann, K. Lingemann, J. Hertzberg and S. Thrun, “6D SLAM with an application in autonomous mine mapping,” *Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004 IEEE International Conference on*, 2004, pp. 1998-2003 Vol.2.

[4] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[5] NASA’s Ninth Annual Robotic Mining Competition Rules Rubrics, May 14-18, 2018, Kennedy Space Center www.nasa.gov/sites/default/files/atoms/files/2018_rules_rubrics_parti.pdf

PLACE
PHOTO
HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.