Exogenous Fault Detection of Autonomous Mobile Robots to Assist in Motion Planning

Guide: Dr. Carlo Pinciroli
Dept. of Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA
cpinciroli@wpi.edu

1st Pratik Jawahar
Dept. of Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA
pjawahar@wpi.edu

***Abstract*—Motion Planning has advanced significantly with respect to autonomous mobile robots over the years to the point where there are a plethora of algorithms that are getting more accurate with every research article in the field. However, when it comes to implementing these algorithms on real autonomous mobile robots, a lot of other parameters factor in to the success or failure of the algorithm. The most common factors that might affect a motion planning algorithm are faulty sensors and actuators. Thus, there is an evident need for an error analysis algorithm that supports and ensures acceptable implementation of the motion planning algorithm. Such a system is even more important in autonomous robots that operate without human monitoring for the most part. This project explores and provides one such error analysis algorithm that can be used to detect a fault in the robot during operation.**

***Index Terms*—error analysis, autonomous, ground robots, wheeled robots**

I. Introduction

Autonomous Mobile Robots (AMR) have been heavily researched since their inception to this day, primarily because of the myriad possible applications for such robots be it in a search and rescue based operation or for industrial tasks such as warehouse automation. There has been significant development in terms of the major systems that comprise AMRs such as sensing and perception, motion planning, control algorithms and hardware systems. Most of these systems are well developed and capable of being readily deployed for tasks such as warehouse automation.

However, when it comes to operation in a real dynamic environment, hardware components like sensors and actuators can fail without warning due to multiple reasons. Since autonomous robots are developed with the intention of not requiring constant human monitoring, the system needs to be able to detect such faults and errors during operation.

Fault Detection and Error Analysis is not a new field in itself. It has been researched upon in depth for applications of other industrial electro-mechanical devices that operate without constant human monitoring such as HVAC systems. The most common technique used in such fault detection systems for electro-mechanical devices is model based fault detection.

Fault Detection for AMRs however, is a comparatively under-researched field, but is equally important as any other sub-system. The primary motivation of this paper is to address the lack of research work in the area of fault detection for autonomous robotic systems, sto facilitate deploying such systems in real dynamic environments.

II. Related Work

Fault Detection systems have been in operation since the advent of automation in electro-mechanical devices, with the earliest examples dating back to the early 1980s. The increasing complexity of devices and systems with increasing levels of autonomy perpetuates the need for such systems that preserve the safety of the device itself and its operating environment. The earliest methods involved using additional proprioceptive sensors and hard-coding limits of normal operation. This ensured that the sensors detected common faults that were either recurrent or predictable in terms of whether they could occur and the detection system itself only detects when that particular fault occurred. These were common subsystems during the rise of the aviation industry. However, they became obsolete when considering sensor data reliability.

The next advancement came when a list of model based fault detection systems were developed. These systems use a modelling technique to encapsulate the normal operating state of the system, and comparing this with the instantaneous state of operation at each time step [5]. This enables the fault detection system to pick out multiple errors without the need to hard-code each error individually.

The most significant advancement in fault detection systems came with the use of particle filters to study and monitor system operation [3] [7]. These filters employ probabilistic models to assign a probability to the normal functioning of each sub-system based on certain parameters. Combining these using conditional probability based algorithms, an efficient real-time system can be built to detect fault or anomalies in system operation. Such systems are commonly used in space missions like the Mars Rover mission by NASA, or on the International Space Station to monitor systems that cannot be monitored by humans continually.

Recent advancements in this field employ the use of Machine Learning techniques to classify system operation as normal or anomalous by building a model to learn the systems normal states of operation [6]. Most commonly used

techniques involve classification using Support Vector Machines (SVMs) or building Artificial Neural Networks such as Multi-Layered Perceptrons (MLP) that learn more accurate representations of the system and can thus be used to identify faults accurately and in real time.

III. Methodology

A. Objective

The goal of this paper is to build a Fault Detection system for an autonomous robot, such that locomotion based tasks such as motion planning and trajectory tracking can be performed in a dynamic environment.

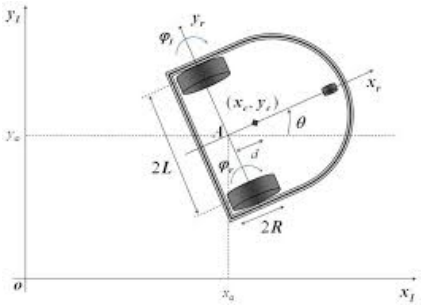

Fig. 1. Differential Drive Robot [4]

B. System Model

1) Kinematic Model: The Kinematic Model represents the body velocity of a system in terms of its joint velocities. In our case, the body velocity is the velocity of the Center of Mass of the robot and the joint velocities are the wheel velocities.

$$\begin{bmatrix} \dot{x}_a^r \\ \dot{y}_a^r \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ \frac{R}{2L} & \frac{-R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix} \tag{1}$$

2) Kinematic Constraints: The kinematic constraints for the differential drive robot are obtained from two non-holonomic constraints, no lateral slip condition and pure rolling constraint [4]. This means the wheels are always in pure rolling and there is no sliding, and the robot cannot slip directly along the direction of its axle.

$$\Lambda(q)\dot{q} = 0 \tag{2}$$

where,

$$\Lambda(q) = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 & 0 \\ \cos\theta & \sin\theta & L & -R & 0 \\ \cos\theta & \sin\theta & -L & 0 & -R \end{bmatrix} \tag{3}$$

and,

$$\dot{q} = \begin{bmatrix} \dot{x}_a^r & \dot{y}_a^r & \dot{\theta} & \dot{\varphi}_R & \dot{\varphi}_L \end{bmatrix}^T \tag{4}$$

3) Dynamic Model: We form the Dynamic Model of the system using the Lagrangian approach that gives the required model shown in Eq.5

$$M(q)\ddot{q} + V(q,\dot{q})\dot{q} + g(q) = B(q)\tau - \Lambda^T(q)\lambda \tag{5}$$

where, $M(q)$ is the inertia matrix, $V(q,\dot{q})$ is the coriolis matrix, $g(q)$ is the gravitational matrix which vanishes since we assume planar motion on the ground, $B(q)$ is the torque input matrix, τ is the net torque on the system, $\Lambda(q)$ is derived from the kinematic constraints mentioned above and λ reprsents the Lagrange multipliers λ_i.

$$M(q) = \begin{bmatrix} m & 0 & -md\sin\theta & 0 & 0 \\ 0 & m & md\cos\theta & 0 & 0 \\ -md\sin\theta & md\cos\theta & I & 0 & 0 \\ 0 & 0 & 0 & I_w & 0 \\ 0 & 0 & 0 & 0 & I_w \end{bmatrix} \tag{6}$$

$$V(q,\dot{q}) = \begin{bmatrix} 0 & -md\dot{\theta}\cos\theta & 0 & 0 & 0 \\ 0 & -md\dot{\theta}\sin\theta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

$$B(q) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}; \lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \end{bmatrix} \tag{8}$$

This dynamic model is used to generate simulations of the robot since its a complete representation of the system dynamics and it is also very useful when it comes to writing control algorithms to make the robot perform desired tasks.

C. Particle Filter Algorithm

The idea of a Particle Filter (PF) is presented as a salient approximation of recursive probabilistic filters used for state estimation, for e.g. Bayesian Filters [7]. State estimation is the process of predicting the state of the system, given a set of sequential information on the system. It is broadly classified into Batch estimation and Recursive estimation. The former considers all sequential data points with equal importance. As a result of this, it is very computationally expensive even for minutely long sequential processes. Recursive estimation on the other hand takes a probabilistic approach with higher importance given to past and future events that are closest to the state to be estimated. In our work, we explore recursive estimation techniques since trajectory tracking is usually a long sequential process.

For this paper we focus on a first order Markov formulation for discrete time state estimation. Consider $X = \{x_1 ... x_t\}$ to be the system states for time $\{1 : t\}$. Let $z_{1:t} = \{z_1 ... z_t\}$ be the state measurements and a_t be the control action for the interval $(t-1, t]$. The aim of the state estimation problem is to

calculate the probability density function (pdf) $p(x_t|z_{1:t}, a_t)$. The Markov formulation of this is given in Eq.9

$$p(x_t|z_{1:t}, a_t) = \eta_t p(z_t|x_t) \int p(x_t|a_t, x_{t-1}) \times p(x_{t-1}|z_{1:t-1}, a_{t-1}) dx_{t-1} \quad (9)$$

where, η_t is a normalizing constant. Here,$p(z_t|x_t)$ is the *observation function*; $p(x_t|z_t, x_{t-1})$ is called the *state transition function*. This formulation assumes that the initial pdf of the system or the *prior pdf* $p(x_0)$ is known. Since there is no closed loop solution to this Markov formulation, we implement a particle filter approximation. A particle filter is essential a Monte Carlo approximation of the posterior distribution in Eq.9 [7].

Particle filters use a fully initialised set of N particles that are formulated as, $\{(x_t^1, w_t^1), ...(x_t^N, w_t^N)\}$ where (x_t^i, w_t^i) is the state and importance weight of particle $'i'$ at time $'t'$. The Monte Carlo approximation of the posterior distribution is then given by,

$$\hat{p_N}(x_t|z_{1:t}, a_t) = \sum_{i=1}^{N} w_t^{[i]} \partial_{x_t^{[i]}}(x_t) \quad (10)$$

where, ∂ is the Dirac Delta Function. It can be shown that as $N \rightarrow \infty$, $Eq.10 \rightarrow Eq.9$, which is the true posterior distribution.

Particles of every subsequent time step are then updated by recursively drawing elements from the approximated posterior distribution. This however is not a very feasible task to do in terms of computational intensity, so we draw from a more easily conceivable approximated distribution $'q()'$ which is called the importance distribution. The discrepancy caused by this approximation is accounted for by using the importance weights associated with each particle, which is given by,

$$w_t^{[i]} = \frac{p(x_t^{[i]}|x_{t-1}^{[i]}, a_t)}{q(x_t^{[i]}|z_{1:t})} \quad (11)$$

The choice for $q()$ determines the verity of the algorithm. Generally, the simplest choice for $q()$ is the transition probability function, which makes the importance weight equal to the observation likelihood function. This forms the algorithm for the Classical Particle Filter (CPF)

1) Degeneracy Problem:: The CPF algorithm faces a recurring issue of degeneracy. After a small number of iterations, the weights of all particles but for one, become negligible, since the variance of the particles in a CPF is bound to increase with time. This problem can't be avoided directly, and it means a lot of computation is wasted in providing negligible weight updates.

A solution to this problem comes from tracking an effective number (N_{eff}) of particles (whose weights are significant), and every time N_{eff} falls below a set threshold (N_T), the particles are resampled with techniques such as resampling with replacement [1]. N_{eff} is calculated as,

$$N_{eff} = \frac{N}{1 + Var(w_t^{*i})} \quad (12)$$

where, w_t^{*i} is the true weight given by Eq11. Since this can't be calculated easily, we make a tangible approximation as,

$$\hat{N_{eff}} = \frac{1}{\sum_{i=1}^{N} (w_t^i)^2} \quad (13)$$

The resampling is then performed according to Algorithm 2. in [1]. The final modified particle filter algorithm is then given by,

Algorithm 1: Modified Particle Filter

Result: $\left[\{x_t^i, w_t^i\}_{i=1}^N\right] = PF\left[\{x_{t-1}^i, w_{t=1}^i\}_{i=1}^N, z_t\right]$
Set $p(x_0)$ to the prior pdf;
Draw intitial particles B_0;
for *i=1:N* **do**
  Draw $x_t^i \sim q(x_t|x_{t-1}^i, a_t)$;
  Assign w_t^i according to Eq.11;
end
Calc. total weight, $W_{tot} = SUM\left[\{w_t^i\}_{i=1}^N\right]$;
for *i=1:N* **do**
  Normalize: $w_t^i = W_{tot}^{-1} w_t^i$
end
Calc. $\hat{N_{eff}}$ from Eq.13;
if $\hat{N_{eff}} < N_T$ **then**
  Resample acc. to Alg.2 in [1]
end

IV. Experimental Results

A. Experiment

The experiment for testing the Fault Detection system will be centered around an autonomous differential drive robot shown in Fig.1 [4]. The robot's goal would be to track a given trajectory. The sub-systems of the robot for our experiment include two actuators (one to drive each wheel), two wheel speed sensors, and an external 2D imaging sensor which gives the ground truth data about the robot states, but has no information about the control inputs the robot receives. Fig.2 shows a single frame from the imaging sensor output. This is the simulation environment for the experiment.

During the trajectory tracking operation, a random type of fault will be injected into the system at a random instant for a random duration, and the fault detection system will be used to detect it. The performance of the detection system will be measured by a metric that accounts for the time lag between the fault being injected and detected.

The types of faults that could be injected individually or together include:

1) Actuator error: The actuator is not able to provide the required torque or provides excess torque causing a change in wheel speed
2) Sensor error: The actual wheel speed and the wheel speed recorded by the sensor do not match

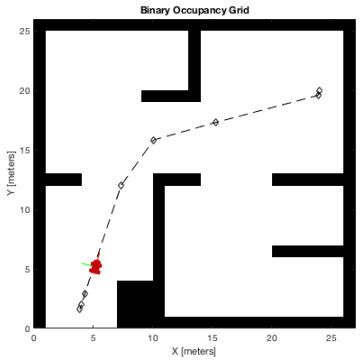

Fig. 2. Simulation Environment

First, based on a given start and end point and the map, a Probabilistic Roadmap (PRM) algorithm is used to construct the required trajectory [2]. Fig.3 represents the working of the algorithm and the extracted trajectory.

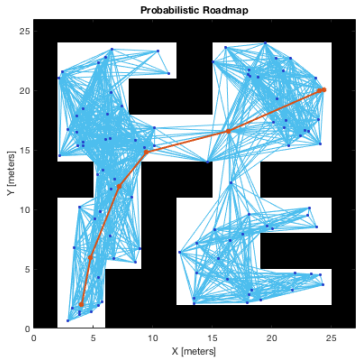

Fig. 3. Probabilistic Roadmap Output

A simple model based trajectory tracking controller is then implemented for the robot to track the extracted trajectory. At a randomly chosen instant and for a randomly chosen duration, a fault is induced to simulate uncontrollable slip, as might be the case if say, that region of the floor were covered in a lubricant or that region might have been a slippery slope and not a flat surface as shown in the 2D map. The proprioceptive measurements however are not changed, which means the robot would assume it is in a normal state of operation. This is a combination of both actuator and sensor error and the goal of the fault detection system here would be to detect the fault as early as possible, despite these complications.

B. Results

The modified particle filter algorithm gives a state estimate at each time step based on previous states, measurements and control inputs. The absolute error between the predicted state and the actual state is assumed to be the metric for this experiment. An error threshold is set and when the absolute state error crosses the threshold, the system is said to have entered a Fault State. The robot is then brought to a halt such that its fault can be diagnosed and fixed. Diagnosis and Fixes are however, out of the scope of this project.

Fig.4 shows the evolution of the absolute state error for a case where no error was injected. It is evident that the error decreases over time meaning the filter gets more confident in state estimations as compared to the estimate based on just the prior pdf.

Fig.5 shows the evolution of the state error with time. Under normal operation the absolute error decreases with time as expected but as soon as the system enters a faulty state, the absolute error value explodes, indicating a fault.

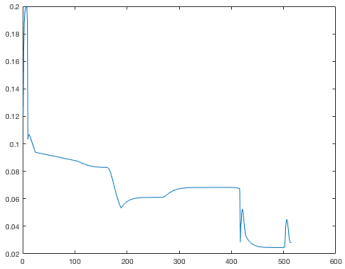

Fig. 4. Evolution of Absolute State Error vs Time

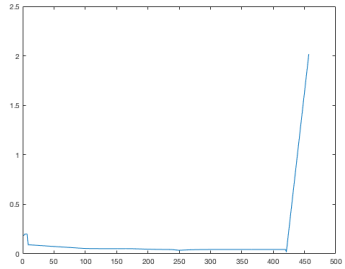

Fig. 5. Evolution of Absolute State Error vs Time

The detection threshold is set at $Err_{Thresh} = 2m$ which is around 10 times the mean values under normal operation. Under this configuration, the system takes just 9 time steps since error injection for the error to cross the threshold. With a sampling rate of 0.1, the fault is detected in $\sim 0.9s$ The average state error value for normal operation is around $0.06m$ and a fault state immediately causes the state error to explode, meaning the threshold can be set to a much lower value than $'2'$ for faster detection. Fig.6 shows the simulation result corresponding to Fig.5.

V. Conclusions

The modified particle filter is an efficient, computationally optimal algorithm for fault detection based on discrete-time state estimation. The more the number of particles used, the more accurate the fault detection system is, implying the

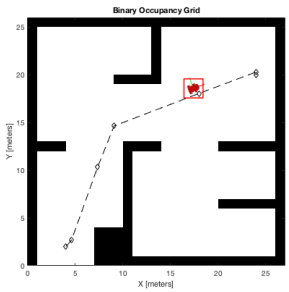

Fig. 6. Simulation Result corresponding to Fig.**??**

lesser the time elapsed in detecting a fault after it has been injected. However, the particle filter itself relies heavily on the system model and can only track a pre-defined set of states. This means for systems whose states cannot be modelled efficiently, the PF algorithm cannot track those states. It would be interesting to study algorithms for fault detection that are not as heavily model dependent as part of future work.

REFERENCES

[1] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[2] Valérie Boor, Mark H Overmars, and A Frank Van Der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 2, pages 1018–1023. IEEE, 1999.

[3] Richard Dearden and Dan Clancy. Particle filters for real-time fault detection in planetary rovers. 2001.

[4] Rached Dhaouadi and A Abu Hatab. Dynamic modelling of differential-drive mobile robots using lagrange and newton-euler methodologies: A unified framework. *Advances in Robotics & Automation*, 2(2):1–7, 2013.

[5] Paul M Frank and Xianchun Ding. Survey of robust residual generation and evaluation methods in observer-based fault detection systems. *Journal of process control*, 7(6):403–424, 1997.

[6] LB Jack and AK Nandi. Fault detection using support vector machines and artificial neural networks, augmented by genetic algorithms. *Mechanical systems and signal processing*, 16(2-3):373–390, 2002.

[7] Vandi Verma. *Tractable particle filters for robot fault diagnosis*. PhD thesis, Stanford University, 2004.