Extended Kalman Filter and Linear Quadratic Regulator on Differential Mobile Robot for Trajectory Tracking

Ahmed Khalil and Mohamed Safwat

***Abstract*— We describe a set of algorithms implemented on a differential mobile robot for improved trajectory tracking. The algorithms improve the single measurement state estimation methods, such as dead reckoning, by integrating an IMU sensor and fusing it with wheel encoders using an Extended Kalman Filter. The project also implements a Linear Quadratic Regulator (LQR) controller for better trajectory tracking performance when compared it to a Proportional Integral Derivative (PID) controller. To aid with tuning parameters and verifying algorithms, various techniques were used to speed up development, such as using simulation and visualization software like Gazebo and rviz. The implementations take advantage of the Robot Operating System (ROS) thereby making it accessible to any differential mobile robot.**

I. INTRODUCTION

Reliable navigation is a challenging field of research in autonomous mobile robotics and it can be split into two categories: indoor and outdoor environments [1]. For effective navigation, the robot must be successful in the four building blocks of navigation: perception, the robot must interpret its sensors to extract meaningful data; localization, the robot must determine its position in the environment; cognition, the robot must decide how to act to achieve its goals; and motion control, the robot must modulate its motor outputs to achieve the desired trajectory [2].

Out of the four navigation components, motion control has received great research attention due to the complex system models and lack of computational power on mobile robots. In general, the dynamics and kinematics of a mobile robot are nonlinear and consist of many uncertainty parameters. Additionally mobile robots are nonholonomic, meaning the controllable degree of freedom is less than the total degrees of freedom [3]. A mobile robot has three degrees of freedom; i.e. its position in two axes and its orientation. However, there are only two controllable degrees of freedom which are the speed of each wheel. An effective controller therefore requires nonlinear and statistical approaches.

This project aimed to tackle the problem of motion control for wheeled mobile robots and improve on the techniques presented in class. Specifically, the PID controller introduced in class will be compared to an Linear Quadratic Regulator (LQR) implementation and the dead-reckoning method discussed in class will be improved on by integrating an inertial measurement unit (IMU) into an Extended Kalman Filter (EKF). Starting from an initial configuration (position and orientation), the goal of the controller is to navigate the robot to reach a pre-defined final configuration. A hybrid approach with the combination of statistical state estimation and optimal control is proposed. First, the kinematics of the robot is reviewed and the configuration variables of the robot are reformulated in the form of navigation variables.

The main contributions of this project are (i) a robust controller for a differential mobile robot system with external disturbances; (ii) an effective statistical state estimation algorithm that incorporates wheel encoders and an IMU; and (iii) open-source code and instructions for implementing all algorithms.

[1]Undergraduate students with the Department of Mechanical Engineering at the University of Wisconsin Madison, Madison WI, USA {ahmed.khalil, safwat}@wisc.edu

II. DIFFERENTIAL ROBOT KINEMATICS

Many mobile robots use a drive mechanism known as differential drive. It consists of 2 drive wheels mounted on a common axis, and each wheel can independently being driven either forward or backward. While the velocity of each wheel can be varied, for the robot to perform rolling motion, the robot must rotate about a point that lies along their common left and right wheel axis. A model of the differential mobile robot, with two rear wheels and a frontal caster wheel as shown in figure 1.

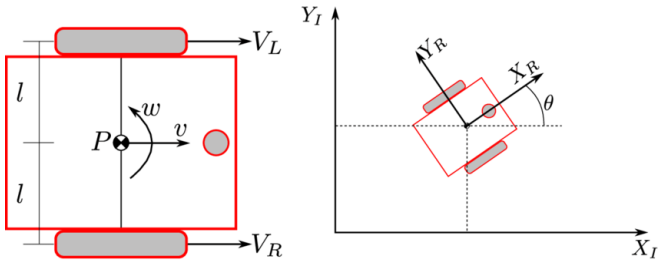

Fig. 1. Ideal differential robot with relative and global coordinates [4]

The rear wheels are actuated by two DC motors, while the caster wheel is not. The motor applied voltage is then directly related to the angular speeds of the wheel by the motor equation,

$$V_{motor} = RI + k_t\omega + L\frac{dI}{dt} \tag{1}$$

Where R is the motor winding resistance in (Ω), I is the current flowing through the motor windings in (A), k_t is the motor constant in ($\frac{V-s}{rad}$), ω is the speed of the motor in ($\frac{rad}{s}$), L is the motor inductance in (H), and $\frac{dI}{dt}$ is the rate of change of current in ($\frac{A}{s}$). The velocity of a single wheel is then given by,

$$v_{wheel} = \omega_{wheel} r \tag{2}$$

Where r is the radius of the wheel. From figure 1, the vehicle's center velocity is given by the following relation,

$$v = \frac{v_r + v_l}{2} \tag{3}$$

Where v_r and v_l are the velocities of the right and left wheels, respectively. The angular velocity of the mobile robot is also related to the left and right wheel velocities, and is given by,

$$\omega = \frac{v_r - v_l}{2l} \tag{4}$$

III. EXTENDED KALMAN FILTER

To account for and filter the sensor measurement noise, a Kalman Filter was used. The Kalman Filter is a recursive algorithm used for estimating the state of the system by considering the measurement and all the noise associated with it, which is typically a Gaussian distribution for linear systems [5]. The algorithm consists of a prediction that was based on the state space model and an update that compared the measurement with the prediction. So, an estimated state in the previous time step along with a current sensor measurement was required to estimate the current state as shown in figure 2.

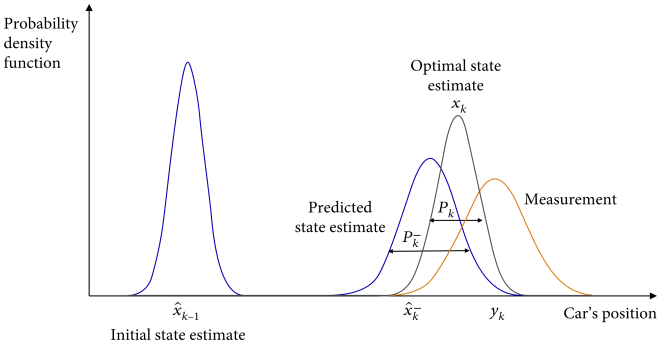

Fig. 2. Kalman Filter for position estimates [5]

A discrete linear space model has the following form:

$$\hat{x}_k = A\hat{x}_{k-1} + Bu_{k-1} + \epsilon_k \tag{5}$$

Where $\hat{x}_k$ is the current estimated state, $\hat{x}_{k-1}$ is the previous estimated state, u_{k-1} is the control input, A is the state transition matrix, B was the input matrix, and ϵ_k is the process noise, which is assumed to be Gaussian with a covariance Q_k, $\epsilon \sim \mathcal{N}(0, Q_k)$. In this case, the state of the mobile robot is given by:

$$\hat{x}_k = \begin{bmatrix} x & y & \theta & v_r & v_l \end{bmatrix}^T \tag{6}$$

The control input are the commanded velocities of the left and right wheels and shown the below:

$$u_{k-1} = \begin{bmatrix} v_{r,k-1} & v_{l,k-1} \end{bmatrix}^T \tag{7}$$

The state transition matrix, A, is as follows:

$$A = \begin{bmatrix} 1 & 0 & 0 & -\frac{1}{2}\sin(\theta_{k-1})dt & -\frac{1}{2}\sin(\theta_{k-1})dt \\ 0 & 1 & 0 & \frac{1}{2}\cos(\theta_{k-1})dt & \frac{1}{2}\cos(\theta_{k-1})dt \\ 0 & 0 & 1 & \frac{dt}{l} & -\frac{dt}{l} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The control input matrix relates the commanded velocities to the state of the robot, and is given by:

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{9}$$

In the update step of the Kalman filter, an observation model was derived. The observation model relates the predicted sensor measurements given estimated state of the robot, and is given by:

$$\hat{y}_k = H\hat{x}_k + \epsilon_k^{'} \tag{10}$$

Where $\hat{y}_k$ is the observation, H is the state to observation transition matrix, and $\epsilon_k^{'}$ was Gaussian noise with covariance $R_k, \epsilon_k^{'} \sim \mathcal{N}(0, R_k)$. In our case, the observation is from the encoder measurements of each wheel, and the yaw angle from the gyroscope. Therefore, the state to observation matrix for the encoders is given by,

$$H_{enc} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

And for the gyroscope,

$$H_{gyr} = \begin{bmatrix} 0 & 0 & 0 & \frac{dt}{l} & -\frac{dt}{l} \end{bmatrix} \tag{12}$$

IV. LINEAR QUADRATIC REGULATOR

With the improved state estimation due to implementing an Extended Kalman Filter, a robust controller was needed to improve the robot's trajectory tracking abilities. We decided to incorporate a Linear Quadratic Regulator (LQR) controller due to it being relatively simple to implement yet very powerful. An LQR controller is a full state feedback controller, with the same structure as a Pole Placement controller without the need to pick pole locations. The LQR controller simply finds the optimal gain, K, by optimizing between system performance and system effort. For this project, the system performance was defined as the robot's state error from the desired trajectory while the system effort was defined as the robot's motor output. The purpose of the LQR controller in this project is to compute velocity of the wheels on the mobile differential robot. The following is an explanation of how the LQR was set up for this project.

The discrete state space model for this differential mobile robot has the same form as in Equation (5), however, the state matrix, A, is a 3x3 identity matrix and the input matrix, B, is defined as following:

$$B = \begin{bmatrix} \cos(\gamma_{t-1}) * dt & 0 \\ \sin(\gamma_{t-1}) * dt & 0 \\ 0 & dt \end{bmatrix} \tag{13}$$

Where γ is the yaw angle of the mobile robot. The state of the mobile robot is defined as:

$$x = \begin{bmatrix} x_{t-1} & y_{y-1} & \gamma_{t-1} \end{bmatrix}^T \tag{14}$$

The control inputs are the robot's center velocity and angular velocity as shown in Equation 15

$$u = \begin{bmatrix} v_{t-1} & \gamma_{t-1} \end{bmatrix}^T \tag{15}$$

With the full space state equation of the robot shown in Equation 16

$$\begin{bmatrix} x_t \\ y_t \\ \gamma_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \gamma_{t-1} \end{bmatrix} + \begin{bmatrix} \cos(\gamma_{t-1}) * dt & 0 \\ \sin(\gamma_{t-1}) * dt & 0 \\ 0 & dt \end{bmatrix} \begin{bmatrix} v_{t-1} \\ \omega_{t-1} \end{bmatrix} \tag{16}$$

With the full space state equation defined, the next step is to solve for the optimal solution by iteratively solving the Infinite-horizon, Discrete time Algebraic Riccati equation (DARE)

$$P = A^T PA - (A^T PB)(R + B^T PB)^{-1}(B^T PA + N^T) + Q \tag{17}$$

And terminating after a maximum number of iterations or after the positive definite solution satisfies a certain tolerance, whichever happens first. The Q matrix defines the weights on the states and will affect the robot's state error. Once the optimal P matrix has been found, the LQR gain is computed using Equation 18.

$$K = (B^T PB + R)^{-1}(B^T PA) \tag{18}$$

Where the R matrix defines the weights on the control input in the cost function and will affect the robot's wheel commands. The optimal control inputs are then calculated using equation 19.

$$u_{star} = -Kx_{error} \tag{19}$$

Where x_{error} is defined as the difference between the actual state and the desired state, as shown below.

$$x_{error} = x_{actualstate} - x_{desiredstate} \tag{20}$$

V. SYSTEM DESIGN

A. Block Diagram

With the state estimator and the controller designed, figure 3 displays the block diagram for the overall system. The block diagram takes into account disturbances and sensor delays.

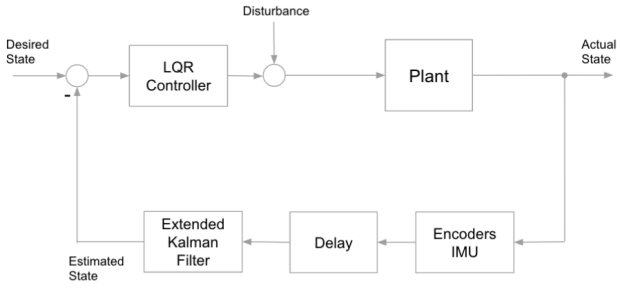

Fig. 3. Closed-loop block diagram for system

B. Trajectory Generation

The desired state in the block diagram is the closest point to the robot on a predefined trajectory. The trajectories used were all cubic functions that were fitted on points specified by us. The algorithms were tested on the following trajectory, which had points of [(0,0), (0,1), (1,0)]. The trajectory is shown in figure 4.

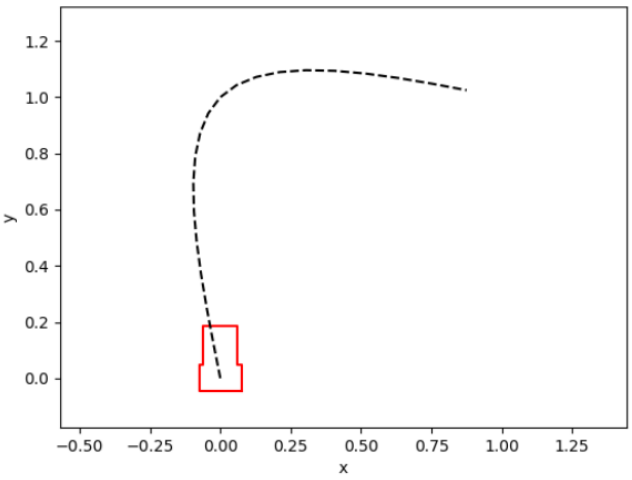

Fig. 4. Trajectory used for testing the algorithm

The trajectories were all cubic functions as mobile robots tend to have finer control over such trajectories [6].

C. System Hardware

The main computer on the mobile robot is a Rasberry Pi with 4GB RAM. The wheel encoders used where the Pololu 3081 Encoder kit and the wheels were the Pololu 1423 60mm wheels. Two IMUs were tested, the Adafruit MPU6050 which has 6 DOF and the Adafruit BNO055 which has 9 DOF. The caster wheel was the Pololu 2691 Ball Caster. A photo of the assembled mobile robot is shown in figure 5.

Given the Raspberry Pi's relatively low processing power, this project was set up to be off-board control with the Raspberry Pi only being used to read IMU and encoder sensor data. This project was run using ROS Networking between a laptop running Linux and the Raspberry Pi. Each

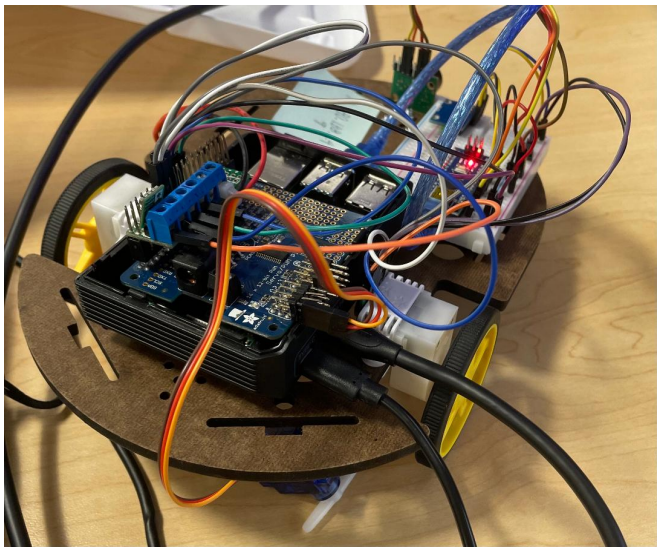

Fig. 5. Hardware on the physical mobile robot

device had its own launch file, with the laptop running the controller.launch file and the Raspberry Pi running the plant.launch file.

D. System Software

The entire project was done on ROS, where the hardware was tested on ROS1 and the simulations were carried out on ROS2. Figure 6 displays the rqt_graph of the project, detailing which nodes were being run on-board and which were being run off-board.

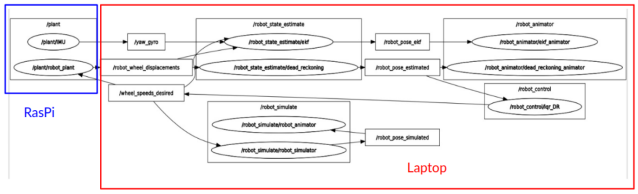

Fig. 6. rqt_graph of project running on the physical mobile robot

Note that this screenshot was taken when the state estimator being used was the dead reckoning. To change the state estimator, the /robot_controller/lqr_DR node can subscribe to /robot_pose_ekf instead of /robot_pose_estimated.

The BNO055 and MPU6050 IMU data were read using Adafruit's CircuitPython BNO055 library and Adafruit's CircuitPython MPU6050 library, respectively. This was significantly faster and less prone to crashing when compared to using the Arduino's serial port.

In order to test the safety and robustness of the LQR controller, a simulation was done in Gazebo, which is a 3D physics simulator that includes a physical model of the robot and an environment. It also came with a differential robot drive plugin that allowed us to subscribe to the odometry topic and publish to the control inputs of the robot. The mobile robot was modeled using a Simulation Description Format (SDF) that included a body, 2 wheels, and a caster wheel as shown in figure 7.

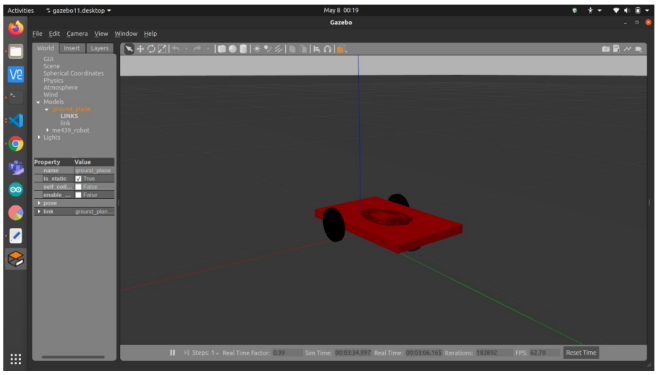

Fig. 7. Differential Mobile Robot SDF in Gazebo

Finally, Gazebo was integrated with ROS2 with the available ros/gazebo package and the communication between ROS1 and ROS2 was done using a network bridge package called ros1_bridge.

VI. Video Demonstrations

Demonstrations of the hardware tests and simulations can be found in the following YouTube videos:

- EKF and LQR code in ROS1 on hardware: https://youtu.be/iVwEiv4ZqnQ
- LQR Simulation in ROS2 and Gazebo: https://youtu.be/bsZWqP-vrrM

VII. Conclusions

From testing on the basic trajectory mentioned earlier, the LQR and EKF algorithms showed significant improvement over PID and dead reckoning. Aditionally, development of software for simulating the mobile robot allows others to test their algorithms without the requirement of hardware. In the future, we would like to test our algorithms with more complex trajectories as well as implementing a Madgwick/Complementary filter to account for the gyroscope drifting.

References

[1] A. De Luca, G. Oriolo, and C. Samson. *Feedback control of a nonholonomic car-like robot*, pages 171–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[2] Roland Y. Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. Introduction to autonomous mobile robots. 2004.

[3] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, USA, 1st edition, 2017.

[4] Jose Filho, Elyson Carvalho, Lucas Molina, and Eduardo Freire. The impact of parametric uncertainties on mobile robots velocities and pose estimation. *IEEE Access*, PP:1–1, 05 2019.

[5] Claudio Urrea and Rayko Agramonte. Kalman filter: Historical overview and review of its use in robotics 60 years after its creation. *Journal of Sensors*, 2021:9674015, Sep 2021.

[6] B. Nagy and A. Kelly. Trajectory generation for car-like robots using cubic curvature polynomials. In *Proceedings of 3rd International Conference on Field and Service Robotics (FSR '01)*, pages 479 – 490, June 2001.