12-8-2022

Autonomous Gyroscopic 2-Wheel Differential Robot

Haruka Kido
haruka.kido@und.edu

James Vrtis
james.vrtis@und.edu

Luke Anderson
luke.anderson@und.edu

Tarek Elderini
University of North Dakota, tarek.elderini@und.edu

Autonomous Gyroscopic 2-Wheel Differential Robot

Haruka Kido
School of Electrical Engineering & Computer Science
University of North Dakota
Grand Forks, ND, USA
haruka.kido@und.edu

James Vrtis
School of Electrical Engineering & Computer Science
University of North Dakota
Grand Forks, ND, USA
james.vrtis@und.edu

Luke Anderson
School of Electrical Engineering & Computer Science
University of North Dakota
Grand Forks, ND, USA
luke.anderson@und.edu

Dr. Tarek Elderini
School of Electrical Engineering & Computer Science
University of North Dakota
Grand Forks, ND, USA
tarek.elderini@und.edu

***Abstract*—This paper demonstrates the implementation of an autonomous gyroscopic 2-wheel differential robot, including a forward and inverse kinematics simulation in MatLAB, a test hardware robot and programmed demonstration of a simple move forward and spin left motion, and a final configuration of a complete square path based on programming kinematics, gyroscopic speed responses, and remote-control functionality.**

Keywords—autonomous robot, gyroscopic control, differential drive, differential drive robot, waypoint tracking

I. Introduction

This report outlines a differential drive robot autonomous motion. The differential drive kinematics analyzes how a mathematical model of forward and inverse kinematics to perform an autonomous functioning robot. In a differential-drive vehicle there are two fixed wheels with a common axis of rotation, and one or more caster wheels. The two fixed wheels are separately controlled, while the caster wheel is passive [5]. The creation of the simulation model was redesigned and built with real parts.

II. Mathematical Model

The mathematical format for this model uses forward and inverse kinematics for autonomous robotic control. To calculate the forward kinematics of a differential drive robot, the following variables are needed: L (length between wheels to center), r (diameter of wheels), θ (pose orientation), & φ_1 / vφ_2 (speed of the right /left wheels) [8]. The right and left wheel positions are calculated using Equation 1 and Equation 2 respectively [8]. The left and right wheel rotation velocity of the robot is calculated using Equation 3 and Equation 4 respectively [8]. The pose of the robot is found using Equation 5.

$$x_{r1}(right\ wheel\ position) = \frac{r\varphi_1}{2} \tag{1}$$

$$x_{r2}(left\ wheel\ position) = \frac{r\varphi_2}{2} \tag{2}$$

$$\omega_1(right\ wheel\ rotation\ velocity) = \frac{r\varphi_1}{2*l} \tag{3}$$

$$\omega_2(left\ wheel\ rotoation\ velocity) = -\frac{r\varphi_2}{2*l} \tag{4}$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \frac{r_R}{2} * cos\theta & \frac{r_L}{2} * cos\theta \\ \frac{r_R}{2} * sin\theta & \frac{r_L}{2} * sin\theta \\ \frac{r_R}{2L} & -\frac{r_R}{2L} \end{bmatrix} * \begin{bmatrix} \varphi_1 \\ \varphi_2 \end{bmatrix} \tag{5}$$

III. Simulation Work

The MatLAB simulation includes a waypoint path simulation using the Jacobian matrix. Using forward kinematics, the algorithm determines position and orientation of the object given the values of the coordinate variables. Using inverse kinematics, the algorithm determines the subsequent coordinate displacements using the current position and orientation of the object.

Figure 1 graphically displays the different parameters the robot needs with respect to the reference frame. The η is the velocity vector/pose for the center of the robot; containing *x*, *y*, and Θ, which respectively are the distance and angular displacement of the robot. The *u* and *r* variables are the forward and angular velocities, respectively [1].

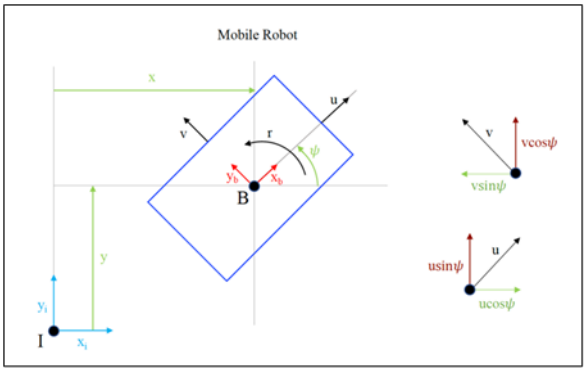

Fig. 1. Mobile robot reference frame [1]

The robot waypoint tracking method is found in Figure 2. The variable ρ in Equation 7 calculate the distance between the desired pose and current pose. Equation 8 and Equation 9 calculate the angular displacement of the robot. Equation 10 calculates the angular displacement error between η the existing pose, once the margin of error is below a specified tolerance. Equation 11 is the rotation matrix vector by angle θ about the z-axis. Found in Equation 12.a is where the matrix controller adjusts the forward and angular velocities and is dependent upon the robot platform parameters in Equation 12.a [1].

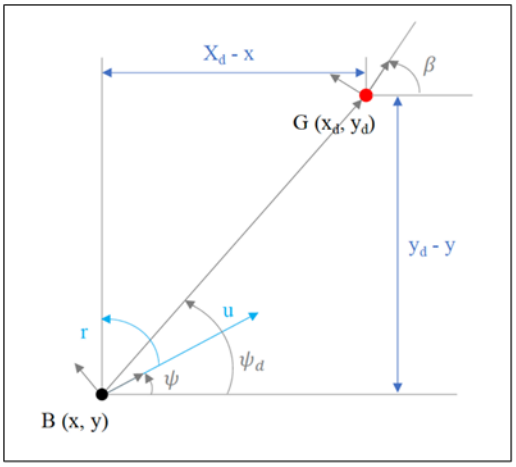

Fig. 2. Mobile robot line of site reference frame [1]

$$\dot{\eta} = \begin{bmatrix} x_d \\ y_d \\ \theta_d \end{bmatrix} \tag{6}$$

$$\rho = \sqrt{(\dot{\eta}(x) - \eta(x,t))^2 + (\dot{\eta}(y) - \eta(y,t))^2} \tag{7}$$

$$\theta_d = \text{atan}(\tfrac{y_d - y}{x_d - x})(t) \tag{8}$$

$$e_{error} = \begin{bmatrix} x_d \\ y_d \\ \theta_d(t) \end{bmatrix} \tag{9}$$

$$e_{radius-error} = \dot{\eta} - \eta(t) \tag{10}$$

$$J = \begin{bmatrix} cos\theta(t) & -sin\theta(t) & 0 \\ sin\theta(t) & cos\theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

Kinematics Velocity Controller

$$\begin{bmatrix} u(t) \\ 0 \\ r(t) \end{bmatrix} = Z = J^{-1} * \left(\begin{bmatrix} K_{vel-x} & 0 & 0 \\ 0 & K_{vel-y} & 0 \\ 0 & 0 & K_{vel-\theta} \end{bmatrix} * \begin{bmatrix} x_d - x(t) \\ y_d - y(t) \\ \theta_d - \theta(t) \end{bmatrix} \right) \tag{12.a}$$

$$\begin{bmatrix} u_z(t) \\ 0 \\ r_z(t) \end{bmatrix} = \left(\begin{bmatrix} a/2 & a/2 \\ 0 & 0 \\ \frac{a}{2*L} & -\frac{a}{2*L} \end{bmatrix} * \begin{bmatrix} u(t) \\ 0 \\ r(t) \end{bmatrix} \right) \tag{12.b}$$

Position Controller

$$\eta(:, t+1) = \eta(t) + \int_t^{?} error(t)\, dt = 0 \tag{13}$$

Constants: a (wheel diameter), d (between wheels and center), l (robot length), $\dot{\eta}$ (desired position and orientation), $\eta(0_i)$, K_{vel}

Loop

For $0_i : n_i$
ρ(t)
$e_{error}(t)$
If ρ(t) <= allowable distance
$e_{radius-error}$ (t)
End
J(t)
υ(t), r(t)
η(t+1)
End

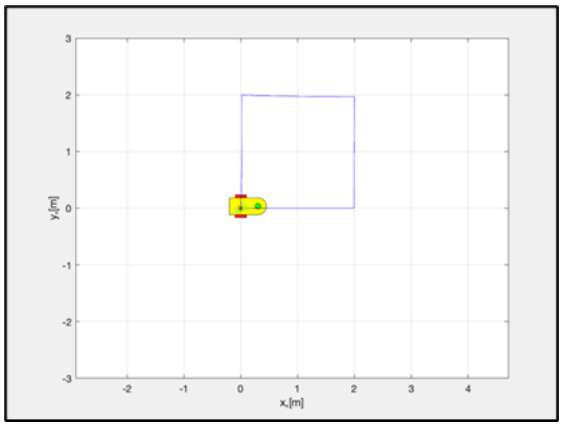

Fig 3. Graphical model of robot path

Figure 4 and 5 contain the graphical representation of the x and y-axis desired endpoints compared to the actual path taken by the robot simulation. The simulation will continue the loop to find the desired location until ρ is less than or equal to provided margin; however, the performance of the simulation may provide unrealistic results if the margins are too restrictive.

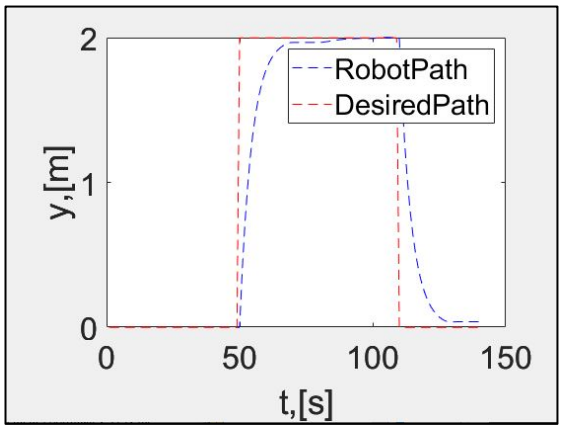

Fig 4. Graphical model of y-axis robot path versus time

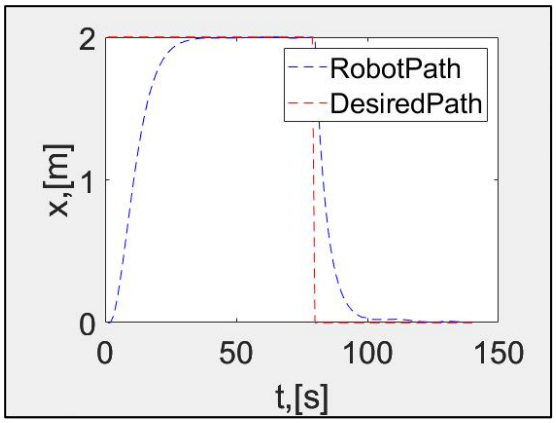

Fig 5. Graphical model of x-axis robot path versus time

IV. Experimental Work

The hardware implementation requires experimental configuration of the 2-wheeled autonomous gyroscopic differential drive robot using a programmable Arduino UNO R3. We build a 2WD differential robot to run a preliminary test of the differential drive script enabled to configure motion control to 2 encoder-configured TTL motors from L298N dual H-Bridge motor drivers with methods including move forward, move back, spin right, and spin left with 2 back motors and 1 front castor baller. Optocoupler isolators as speed sensors are configured to measure the speed of the robot.

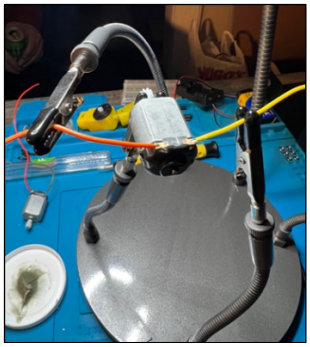

Fig 6. TT DC motors soldered to electrical wires

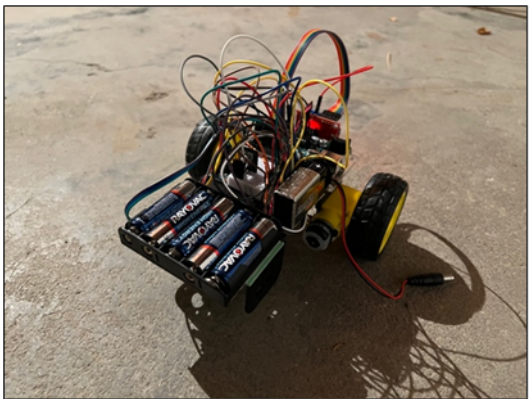

Fig 7. 2WD autonomous differential test robot

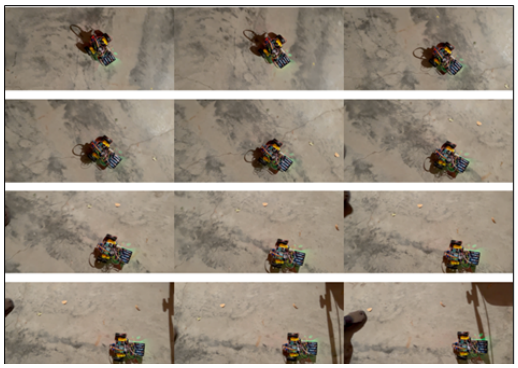

Fig 8. Time Lapse of 2WD differential test robot in forward movement and spin left movement

The preliminary hardware implementation of the test robot was successfully programmed with move forward, move reverse, spin right, and spin left, after attachment of the interrupts to their ISRs. The test motor movement was enabled by scripting experimental sequences that push step count and speed from 0-255 for each kinematic operational function. We then apply the IR remote control functionality and gyroscopic velocity and motion control for the final robot configuration. The circuit wiring diagram of the robot hardware including the motor driver, motors, encoders, speed sensors, batteries, IR sensor, gyroscope, and Arduino UNO R3 is shown in Figure 9 below:

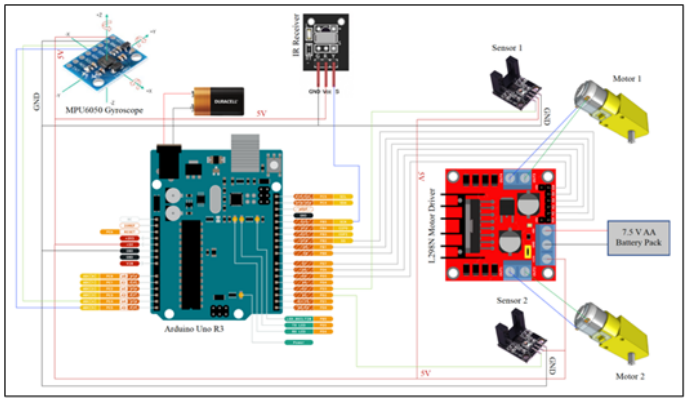

Fig 9. Robot hardware circuit schematic diagram

To achieve the automation for the robot completing the square simulation, we had to make use of the MPU6050 gyroscope and encoders. In order to achieve this, we utilized Arduino libraries and examples from Jarzebski and William Gate. Jarzebski created some good examples for using the MPU6050 gyroscope, and William Gate has an in-depth tutorial on how to implement the encoders that use the LM393 Comparator.

Figuring out how to use the MPU6050 was not the easiest, as the examples provided by Jarzebski lacked good comments on how the example codes worked but eventually, we found a code that could make use of the YAW output provided from the

MPU6050. Using this we could set a yaw limit. For our case this value would be 90°. First, we set the YAW value to 0 every time before we would make a rotation. Next, we made a left spin function stating that would turn the robot left only while the YAW value was less than 90. When the YAW value reached 90°, the robot motors were stopped.

```
// Autonomous "SpinLeft" function ////////////////////////////////////////////////////////////////////////////
void SpinLeft(int steps, int mspeed){                    //sets up "SpinLeft()" function
  counter_A = 0;                                         //reset counter A to zero
  counter_B = 0;                                         //reset counter B to zero
  yaw = 0;                                               //reset yaw = 0
    digitalWrite(in1,LOW);                               //sets in1 LOW
    digitalWrite(in2,HIGH);                              //sets in2 HIGH
    digitalWrite(in3,HIGH);                              //sets in3 HIGH
    digitalWrite(in4,LOW);                               //sets in4 LOW

  while( yaw<90 && steps > counter_A && steps > counter_B){  //while: yaw is less than 90, steps is greater than counter_A ,and steps is greater than counter_B perform th
  gyro();                                                //calls on gyro() function
  if (steps > counter_A) {                               //if steps > counter_A the motor A moves ,else stop motor
  analogWrite(enA, mspeed);
  } else {
  analogWrite(enA, 0);
  }
  if (steps > counter_B) {                               //if steps > counter_B the motor B moves ,else stop motor
  analogWrite(enB, mspeed);
  } else {
  analogWrite(enB, 0);
  }
}
// Stop when done
analogWrite(enA, 0);                                     //stops both motors
analogWrite(enB, 0);
counter_A = 0;                                           //  reset counter A to zero
counter_B = 0;                                           //  reset counter B to zero
}
```

Fig 10. Robot turning function code

The code shown in Figure 11 executed the turning function of our car. The next step to figure out was how to measure the displacement of the car moving forward. For this, we utilized code from William Gate who is also a YouTuber who gives very good (although sometimes exceedingly long) tutorials on the mechanism of encoders for robots that use TT motors. Following his example, we converted the steps of the encoder into a distance value to track the distance the car moved.

```
int CMtoSteps(float cm) {
  int result;                                            // Final calculation result
  float circumference = (wheeldiameter * 3.14) / 10;     // Calculate wheel circumference in cm
  float cm_step = circumference / stepcount;             // CM per Step

  float f_result = cm / cm_step;                         // Calculate result as a float
  result = (int) f_result;                               // Convert to an integer (note this is NOT rounded)

  return result;                                         // End and return result
```

Fig 11. Robot controller steps

With his example, we then created autonomous code like the one previously seen in spin left function. We then created a simple autonomous loop that would act like a startup sequence as soon as the robot was powered on.

To achieve remote control, we decided to go with using an IR remote or in this case a TV remote from a Samsung DVD player. We chose this remote because the IR LED in the DVD remote is much stronger than the LED that we get from our Arduino kits, meaning that we can use it from a good distance. We did try control the robot using radio frequencies using the nRF24L01 and was successful however there seems to be some lag in the code that was not ideal to use. As we plan to use the robot indoors and the IR option takes up less space on the Arduino, it was decided that IR was the best option. For the IR code, we took advantage of the IR remote library and example code from Ken Shirriff, which can take the blinking pattern from an LED source and translate it into a binary which is then translated into a unique hexadecimal number. Using these unique numbers, we created "if" and "else" statements that would perform functions for certain movements such as forward, reverse, right and left. The final robot configuration implementing the autonomous and remote-controllable square path is shown below in Figure 12.

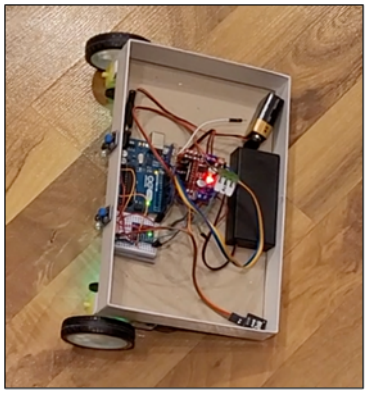

Fig 12. Autonomous gyroscopic robot configuration for square path implementation

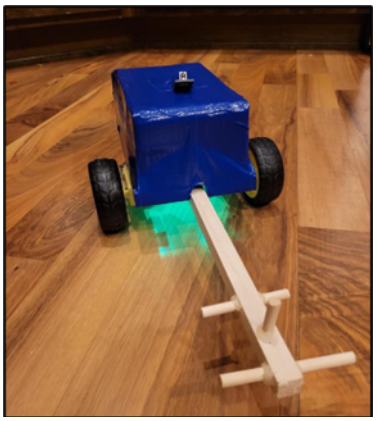

Fig 13. Final robot configuration with fighting tail and covered electrical parts

V. Discussion

Simulations: The simulation model had an error with the line-of-sight method used to perform the autonomous motion. The error would occur when $\dot{\eta}$ contained a value that was previously used in a preceding desired destination. The simulation error is corrected when the robot aligns itself in the negative $\dot{\theta}$ orientation. The error may be caused by the mathematical line of site method to find the robot's path from the starting location to the endpoint location.

Hardware Implementations and Test Drives: The test hardware implementation, possibly due to electrical conductivity insufficiencies, power consumption issues, or motor driver inefficiencies, did not function to continue the motion drive in any direction after the first implementation. Researching the programming requirements as far as viable initiate states and adjustments to meet motion criteria can take the iterative approach, with obvious benefits from having

experience in fundamental computer science concepts such as block code structure requirements, syntax, and acceptable logic.

VI. Conclusion

Robotics with Arduino makes peripheral interfacing accessible and flexible but at the cost of power consumption sensitivities or inaccuracies. The physics of parts' electrical characteristics and material boundary conditions make Robotics a challenging subject between computer science, electrical engineering, and applied material science.

VII. Future Works

To improve the simulation, researching the previously mentioned error when $\dot{\eta}$ contains a value used in a preceding desired destination needs a better understanding. Additionally, the mathematical model needs improvement by adjusting the velocity and computational time for a loop needed for the robot to move from the starting to the ending point. To improve the drive test, it would be worth implementing the test hardware system using all new parts rather than using some old parts and in a configuration that relies on motors that have pre-soldered wire leads so that electrical conductance is at optimum quality. Adding machine vision for image-processing in an autonomous robot can also enhance the capabilities of the differential drive remote-controlled robot.

VIII. References

[1] A. Thondiyath and S. Mohan, "Wheeled Mobile Robots," Indian Institute of Technology, Palakkad.

[2] Siciliano, Bruno, Sciavicco, Lorenzo, Villani, Luigi, Oriolo, Giuseppe, "Robotics Modelling, Planning and Control," Springer, 2009, ISBN: 978-1-84628-642-1

[3] Warren, John-David, Adams, Josh, Molle, Harald, "Arduino Robotics," Technology in Action, 2011

[4] Edwards, Stephen A., "Assembly Instructions for a Motor Robot Car Kit 2WD, L298N Motor driver, HC-SC04 Ultrasonic module, Arduino," July 2018

[5] B. Siciliano, L. Sciavicco, L. Villani and G. Oriolo, "Robotics Modelling, Planning and Control," Springer, 2009.

[6] "Arduino UNO R3," Arduino.cc, https://docs.arduino.cc/hardware/uno-rev3

[7] Muir, Patrick F., Neuman Charles P., "Kinematic Modeling of Wheeled Mobile Robots," Department of Electrical and Computer Engineering, The Robotics Institute, Carnegie-Mellon University, 1986.

[8] Siegwart, Roland., Nourbakhsh Illah R. "Introduction to Autonomous Mobile Robots" Massachusetts Institute of Technology, 2004.