Experimental Validation of a ground robot simulation model during line following task

Yiannis Papelis, Ph.D.
Virginia Modeling Analysis & Simulation Center, Old Dominion University
Norfolk, VA USA
ypapelis@odu.edu

Ntiana Sakioti
Dept. of Electrical & Computer Engineering, College of Engineering, Old Dominion University
Norfolk, VA USA
nsaki001@odu.edu

ABSTRACT

This paper describes a comparative study of the performance of a simulation model of a two-wheeled robot performing a line following tasks versus a similarly setup simulated environment. The robot model includes closed loop kinematics for the chassis motion and a dynamic model of the wheel motors and control algorithm that generate wheel velocity based on actuation commands. The purpose of this study is to evaluate the fidelity of the model as well as the influence of the motor model, sensor model and simulation time-step on the overall system fidelity while the robot is performing a line following task under a variety of aggressiveness parameters. For simulation purposes, The Autonomous Robotics Modeling & Simulation Environment (ARMSE) is utilized while the Stingray robotic kit from Parallax, equipped with several sensors and additional capabilities, is used as the robotic platform.

ABOUT THE AUTHORS

Yiannis Papelis is a Research Professor at the Virginia Modeling Analysis & Simulation Center, at Old Dominion University. Dr. Papelis obtained a Ph.D. from the University of Iowa, a Master's degree from Purdue and a BS degree from Southern Illinois University, all in Electrical & Computer Engineering. Dr. Papelis' research focuses on unmanned systems, autonomous robotics and use of virtual environments in a wide range of areas such as training, STEM education, design optimization and health-care. His research has been funded by numerous federal agencies as well as industry. Dr. Papelis is a member of the Society for Computer Modeling & Simulation and is currently serving as the Vice President of publications for the society.

Ntiana Sakioti is an undergraduate student at Old Dominion University studying Computer and Electrical Engineering with a minor in Computer Science. She has conducted research in Communications for low-cost implementation of systems using Raspberry Pi and Software Defined Radio platforms. Ntiana Sakioti is a member of the Society of Hispanic Engineers and Society of Women Engineers (SWE) where she holds an officer shadow position.

Experimental Validation of a ground robot simulation model during line following task

Yiannis Papelis, Ph.D.
Virginia Modeling Analysis & Simulation Center, Old Dominion University
Norfolk, VA USA
ypapelis@odu.edu

Ntiana Sakioti
Dept. of Electrical & Computer Engineering, College of Engineering, Old Dominion University
Norfolk, VA USA
nsaki001@odu.edu

INTRODUCTION

Interest in autonomous robotics has been on the rise, fueled partially by interest in self-driving vehicles, drones and humanoid robots. For wheeled robots, the line following behavior is a very popular example to demonstrate basic robot autonomy. A line following robot utilizes a set of downward looking sensors to determine the location of a guiding line. A control loop adjusts the steering and/or speed of the robot so that the line remains centered under the sensor which results in the robot following the line. The line can be implemented either as a white line on a dark surface or a black line on a white surface. Line following robots have a simple structure and require minimal circuitry. Numerous implementations have been developed, ranging from pure hardware to hybrid hardware-software focused designs. The line following behavior is very popular because of the relatively low cost associated with the sensors and also the relative simplicity of the guidance algorithm, something that facilitates learning about all aspects of wheeled robotics. It is also an attractive behavior to use in robot competitions as it presents a reasonable challenge, it is visually interesting and can be surprisingly difficult to fully optimize, especially when a robot has constrained on-board computer resources.

Despite the popularity of line following robots, relatively few studies have used simulation to assess the algorithm's sensitivity to sampling speed, robot controllability and sensor resolution. In that sense, the line following algorithm is a very interesting case study for robot program development because of its dependence on non-linear responses associated with the robot. Unless the line course is intentionally designed to be continuous and utilize low curvature, it is necessary to continuously apply steering and/or speed changes in order to maintain a centered line. As a result, motors operate near their non-linear portion of the envelope and the overall robot behavior is typically non-linear as well. This makes it challenging to develop accurate simulations that can be used to study, calibrate and optimize line following behaviors. At the same time, because line following is a closed-loop behavior, one can reasonably expect that errors in a simulation model would be cancelled out due to the self-correcting nature of the algorithm.

This paper presents results obtained from a quantitative assessment of the simulation model of a robot implementing a line following behavior. The goal of the work is to determine if a first-principles simulation model provides enough fidelity in order to develop and optimize a line following algorithm mostly in simulation. To accomplish this, a physics-based model of a robot chassis, its actuators and sensors was developed. Two variations of the line following behavior were implemented and exercised both on the physical robot and in the simulation. Sample runs of the physical robot were measured and data was compared between the physical robot and simulation to determine the deviation between simulation and physical implementation. A high accuracy motion tracking system was used to ensure that the physical and simulation courses were identical. The same system was also used to collect localization data of the physical robot as a means of comparing its behavior to the simulated robot. Results indicate that the simulation provides adequate accuracy to perform most of the time consuming development, calibration and optimization process in simulation.

The remainder of the paper is organized as follows. Related work is covered immediately below, followed by a description of the robot and the associated simulation environment. The control program used in both the real robot and simulation is then described to provide context to the results. The results are then provided along with discussion and observations. Finally, a conclusion summarizes the results and provides recommendations for future work.

RELATED WORK

There are different implementations to improve the accuracy of line following robots. Hasan et al. (2012) have implemented a closed loop system able to follow a visible line (white or black) or magnetic field with very tight curves. LED is used as a light source and LDR as a light sensor to build the line follower with simple electronics instead of microcontrollers to make it cost effective. The robot presented had a moderate speed of .2 m/s with a minimum turning angle of 110° - meaning any angle less than that will cause the robot to miss the line. For the next step in their research the authors mention sonar and infrared sensors so the robot can perform obstacle avoidance.

The use of IR coding was proposed in Barayyan et al. (2015). The authors presented a robot able to follow a dark colored line only and perform landmark navigation inexpensively. A sensor unit equipped with an IR transmitter and detector sends input to a microcontroller which decodes the actions to be performed and sends that output to the actuator comprised of a DC motor. The robot is also enhanced with artificial landmark navigation to perform a decision when having more than one possible path, using embedded IR coding in a "+" intersection. For future work, the authors describe an implementation of IR coding to identify locations for a more intelligent robot.

Su et al. (2010) proposed and verified an optical sensor calibration procedure using pulse-width-modulation. Three line detection algorithms were compared using a linear platform. It was determined that the weighted average line detection algorithm was the best of the three, the other two being line detection via quadratic interpolation, and line detection via interpolation. An encoder was also devised to keep the information from the racing court. The robot was able to perform line following at a highest speed of 1.3 meters per second and can be used in introductory robot courses.

Dupuis and Parizeau focused on creating a robot simulation using an OpenGL engine to generate a realistic virtual environment (2006). The scene can be created with good accuracy as the experimental scene is artificially controlled using an LCD projector. The authors mentioned the importance of differing processing speeds between the physical and simulated robots, as a slow refresh rate can potentially cause the robot to miss the line. The required simulation time was greatly reduced and the expected simulation behaviors were achieved. Addition of further enhancements will be sought by the authors before implementation on a physical robot.

Webots was used by Almasri et al. (2016) to simulate a line following experiment able to perform collision avoidance with objects of multiple sizes and colors. The Webots simulator developed by Cyberbotics was used to validate the performance of the algorithm developed. The simulations were programmed using Matlab and an e-puck robot and the simulated robot acted as expected – performing sharp turns to avoid obstacles, then returning to its path. This technique can be used in different real-time robotic applications.

EQUIPMENT UTILIZED

Physical Robot

The robot used for this study is a differential drive wheeled robot. Its design originates from the Stingray robot originally produced by Parallax Inc. Even though that particular robot is no longer available from the vendor, its design is rather typical and the chassis remains very useful and appropriately sized for educational robotics. The overall platform has been extensively enhanced and modified and the resulting robot is referred to as the Modified Stingray Robot Platform (MSRP) (see Figure 1). The MSRP is equipped with two brushed DC motors driving the wheel via a 40:1 gear mechanism. A motor driver implementing a full bridge is used to regulate power delivery to the motors via a Pulse Width Modulation (PWM) signal generated by the on-board microcontroller. Each wheel can be driven within a range of -100% (full reverse) to 100% (full forward), in 1% increments. The robot is equipped with two tightly coupled microcontrollers and associated circuitry. They are referred to as the Low Level CPU (LLC) and High Level CPU (HLC). Typically, when

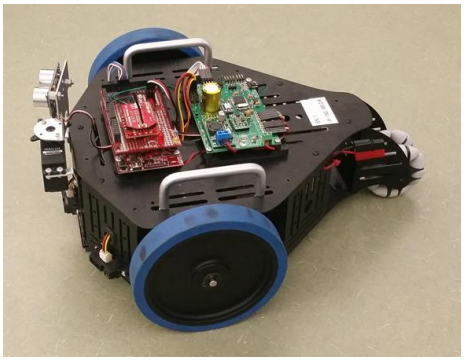

Figure 1 – Depiction of the Robot

programming various algorithms on the robot, the user need only interact with the HLC. The operation of the LLC is transparent as it provides low-level sensor sampling and other 'behind-the-scenes' functionality.

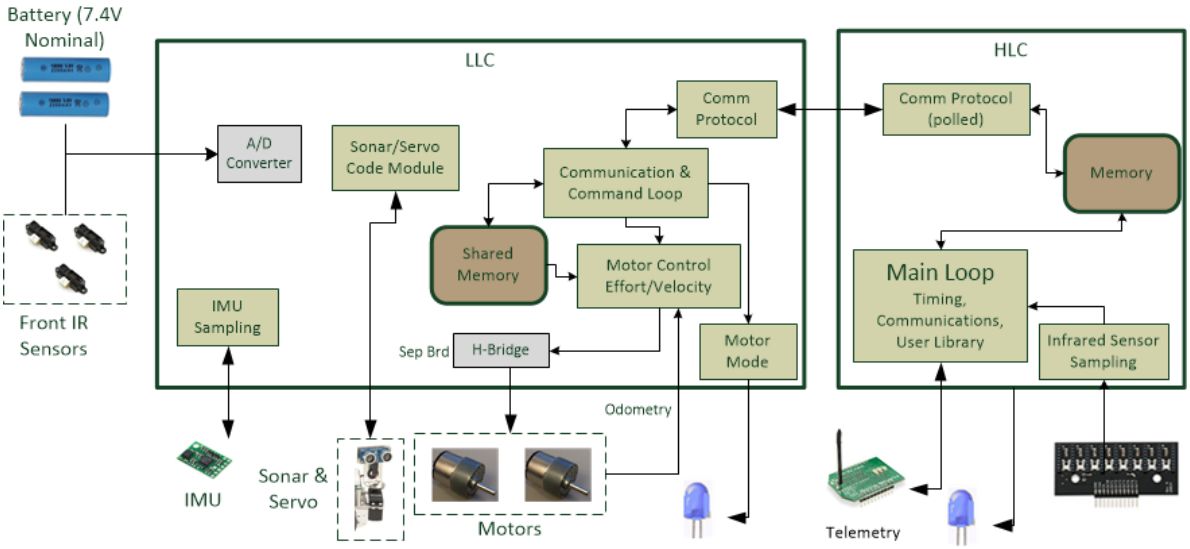

Figure 2 – The Architecture of the Robot

Figure 2 depicts the architecture of the physical robot and apportionment of responsibilities between the LLC and MMC. The LLC is built around the Parallax Propeller chip, a 32 bit, eight core microcontroller with 32 KB of global RAM, 32 KB of EEPROM. Each of the eight cores is clocked at 80 MHz and runs independently of the other cores yet has access to the entire shared memory. The LLC is responsible for generating the PWM signals for the motor driver, sampling the wheel encoders, and generating other signals for controlling and/or sampling on-board accelerometers, solid state gyroscopes and magnetometers, analog proximity sensors as well as controlling the sonar ranging sensor. When controlling the motors, two modes of operation are provided, effort mode and speed mode. In effort mode, the motors are driven open loop, simply by presenting the requested PWM signal to the motor driver. In velocity mode, the control input is desired speed. Feedback from the wheel encoders is used to guide a PID controller implemented on the LLC to seek the desired speed.

The HLC is centered on a 32-bit PIC microcontroller, in particular the PIC32MX795F512 version. It is clocked at 80MHz and is equipped 512KB of program memory and 128KB RAM. It is equipped with Wifi and interfaces with the LLC via a direct low latency serial link. This architecture was selected because it allows the LLC to focus on low level signal manipulation while leaving the HLC free to execute high level programs without worrying about interactions between the low-level control signals and high level algorithms. The HCL issues commands to the LLC to activate sensors and drive the motors (in either speed or effort mode). It can also query the LLC for sensor data. When downloading a program to the robot, the user directly interacts with the HLC which allows programming via a USB link. A thin software layer has been developed to abstract basic operational details and unify the interface between the physical robot and the simulation. The powertrain and on-board electronics are powered by a 7.4V battery. Key mechanical characteristics of the robot are provided in Table 1.

Table 1. Physical Robot Characteristics

Wheel base	13.08981
Wheel diameter	12.3 cm
Max speed	50.3992
Speed Encoders	2080 ticks/revolution sampled @ 25 Hz

Simulation Environment

A Modeling & Simulation (M&S) environment was designed and implemented to support off-line development of robot programs using the MSRP. The M&S environment was initially developed for use in the MSIM 463/563 class, titled "Design and Analysis of Autonomous Robotics Systems" offered by the Modeling Simulation and Visualization Engineering (MSVE) department in the Batten College of Engineering at Old Dominion University.

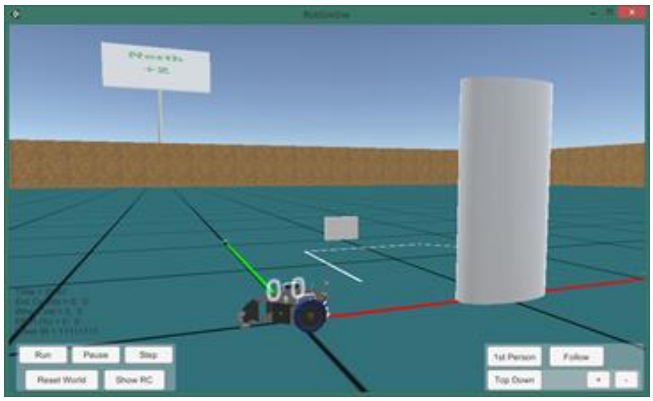

Figure 3 - MSRP Simulation Interface.

The M&S environment provides a simulation of one or more MSRPs in a virtual testbed that can be populated with a variety of objects, obstacles and other features that cause similar responses to the virtual sensors as the actual sensors. Code developed for the MSRP can run virtually unchanged on the robot itself or the simulator, although a different toolchain is used. After developing an MSRP program, it can be uploaded to the physical robot using a USB connection. To use the same program in the simulator, instead of uploading the code to the MSRP's microcontroller, the developer builds a stand-alone executable under Windows that can then communicate with the simulation visualization module that displays the robot's behavior. Of course, all of this is transparent to the user who simply utilizes different tools but the same source for the physical robot and the simulator.

The simulation configuration is driven by a configuration file formatted in XML. The configuration file controls the number of MSRP instances in the simulation as well as the objects that populate the virtual environment. Any number of objects such as arbitrarily sized and rotated rectangles and cylinders can be placed anywhere in the virtual world. In addition, to specifically support line following algorithm development, one can place virtual tape on the simulated floor. To place virtual tape, the user can specify the width of the tape and extend it by providing an arbitrarily long list of vertices.

The MSRP simulation consists of three core modules, the visualization engine, the chassis simulation and the sensors simulation.

Visualization Engine

The visualization engine is built within the Unity3D game development environment. The Visualization engine serves several purposes in addition to providing an interactive visualization of the virtual environment. It provides the main synchronization signal for advancing time in the simulation, facilitates sensor simulation by providing spatial and other queries within the environment and lastly supports data collection.

Upon startup, the visualization engine reads an XML file that describes the virtual world and instances all objects listed in the input file. It then continuously broadcasts a timing signal that can be used to advance the simulation based on fixed time interval. The timing interval is set to 10 mSec, so 100 messages are broadcast each second. Simultaneously with the broadcasting of the timing signal, the visualization engine offers the user an interactive interface that allows pausing the simulation (it simply pauses broadcast of the timing signals) as well as manipulating the view of the simulation. The user can navigate the virtual world in a 1st person view (using the keypad), observe the simulation from a fixed vantage point, or attach the viewpoint to one of the robots in the simulation. In addition, key parameters associated with the robots are displayed. Finally, the visualization engine supports interactive debugging as it displays on a scrolling window all messages printed from inside the robot's program.

Once the visualization engine is loaded, to begin the simulation, the user simply starts executing the self-standing executable that was developed by compiling the robot program. The robot program listens for the timing signal broadcast and executes the user-level code at the same interval utilized on the physical robot. Commands that would affect the robot (such as activating sensors, applying voltage to the motors etc.) are conveyed to the visualization

engine which routes them to the built-in simulations. The results of the simulation are reflected in the movement of the robot (and affected objects) and the resultant state data is made available to the user-level robot program.

Chassis Simulation

The chassis simulation is implemented within the visualization engine. It is centered on a 3rd order model of the electric motor. The model was developed by using the Matlab system identification toolbox to fit a 3rd order linear model whose input is the percentage PWM signal presented to the motor driver and whose output is the wheel rotational velocity. The data used to fit the model was obtained while the robot was driven open-loop and as such already includes the effect of the robot's mass on the response of the motors. Two instances of the model execute for each robot during the simulation, one for the left wheel and one for the right wheel. The derived rotational velocity is used to drive a purely kinematic model of the robot according to the differential drive equations:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t+1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t} + \begin{bmatrix} dt\ \frac{V_L+V_R}{2}\cos(\theta_t + 0.5dt\ \omega_t) \\ dt\ \frac{V_L+V_R}{2}\sin(\theta_t + 0.5dt\ \omega_t) \\ dt\ \omega_t \end{bmatrix} \quad (1)$$

$$\omega = \frac{V_R - V_L}{W} \quad (2)$$

In the above equations, x,y represent the location of the chassis in the inertial (global) coordinate system, and θ is the heading of the robot, dt is the time step used for kinematic estimation, V_L and V_R are the linear velocity of the left and right wheels respectively, ω_t represents the rotational velocity of the chassis at time t and W is the wheel-base. Equations (1) and (2) provide an estimate of the future robot's pose, given the current pose and knowledge of the current wheel velocity, which in turn are generated by the motor model. The visualization engine continuously evaluates this model whose inputs originate in the user-level robot program.

Sensor Simulation

The sensor simulation consists of a series of modules dedicated to each of the robot's sensors. Since the primary interest in this paper is the line following algorithm simulation, only the model of the downward looking sensor is described in detail.

A geometrically accurate virtual model of the robot was developed. Part of the developed geometry includes all sensors on the robot including the downward looking IR sensor. The downward looking IR sensor consists of 8 distinct IR sensors each of which utilizes an IR illuminator and an IR detector. The illuminator emits an IR pulse and the detector outputs a logic value of 1 to indicate detection of the bounced IR signal, or a logic value of 0 to indicate lack of detection. When the sensor is facing a dark floor (or when the robot is more than a few centimeters away from the floor), the IR pulse is not reflected and the detector returns a 0. When the sensor is facing a light colored floor or a light colored tape, the detector returns a 1 to indicate it sees the 'white line'. To simulate this sensor, 8 rays are cast from the geometrical location of the IR emitters in a direction matching the direction of the sensor placement. These 'cast rays', via functionality in the Unity3D gaming engine, return the material associated with the 1st object intersected by the ray. If the brightness of the object exceeds a threshold the sensor registers a logical 1 otherwise the virtual sensor registers a logical 0. This first principle approach to simulating the IR sensor creates a realistic simulation that is fully reactive to the authored virtual environment, including malfunctions in cases where the material is not either pure black or pure white. At the end, designating the floor with a light material and laying down black virtual tape provides functionality identical between the physical robot and virtual environment.

Data Collection Approach

In order to evaluate the applicability of a simulation environment to study line following algorithms, two versions of the algorithm were developed. A line course was also developed for use during testing. The course was originally built manually by placing black tape of 0.02m in width on top of white cardboard. This ensured that the IR sensors could consistently determine the location of the tape relative to the sensor. The width of the tape was such that when under the 8 sensor set, at least two of the IR sensors would be detecting the black tape, provided the tape is within the sensor's range.

Once the physical course was constructed, the motion capture system was utilized to create a replica of the course in the simulator. The motion capture system used for the work described here is built by NaturalPoint and consists of 12 "Flex 3" type IR cameras. Each of these cameras has a resolution of 640x480 pixels and are arranged on vertical tripods surrounding an area of approximately 3.6m x 3.6m. The cameras are equipped with high intensity IR illuminators which allow detection of small markers. After a brief calibration process that registers the location of each camera, the system is capable of tracking at 100 Hz the location of a single marker with sub-millimeter accuracy. In addition, the system can track a rigid body. A rigid body, in this case the robot, is defined by attaching multiple markers to it and using the software to link these markers into a group that can be tracked.

Initially, the course was built by hand in the floor of the laboratory. To re-create the physical course in the simulator, a single marker was used to determine the outline of the tape. The marker was placed on the course and the location estimated by the system was recorded. The marker was then moved approximately two centimeters along the course and the new location was recorded. This process continued until the entire course was mapped. The data was converted into the appropriate XML format so it could be used for describing the course on the simulator. One location on the course was selected to be the origin in both in the virtual and physical course. On the physical course, the origin was established by a special calibration feature of the mocap system. This established the origin which by the nature of the XML file is also the origin of the virtual world. Figure 4 depicts the course as shown via a top-down view in the virtual simulator environment.

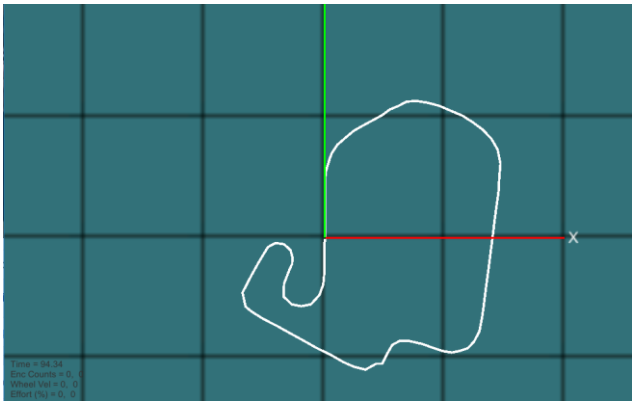

Figure 4 - Line Course in Virtual Environment.

During data collection, the mocap system was also used to track the location of the robot. A set of markers was placed on the robot and the system was trained to identify these markers as a "rigid body". As the physical robot moved about the ground, the mocap system calculated an estimate of the position and orientation of the rigid body and the data was collected into a capture file. At the end of the run, the capture file contained the trajectory of the robot gathered at 100 Hz. The file includes a timestamp, id of all the tracked rigid bodies (only one was present), x, y, z coordinate of the rigid body in millimeters and roll/pitch/yaw estimates. Once the data was collected, a script written in Matlab was used to process and clean up the data. One of the challenges encountered during data collection was that the rigid body heading produced by the mocap system suffered from frequent jumps and reversals. To simplify the calculations, the heading used in the comparison was instead calculated by using trigonometry based on the filtered x and y coordinate. The resultant heading was then "unwrapped" (i.e., when crossing from 2p to 0, a multiple of 2p was added) to ensure a continuous curve. The same processing was made to the heading produced by the simulator.

Two versions of a line following algorithm were used for this experiment. Both versions are effectively implementing a P controller trying to zero the lateral offset of the line as the robot moves along. The lateral offset is calculated by looking at the data returned by the 8-bit downward looking IR sensor. Recall that based on the operation of the sensor, when the line is furthest away to the right, the corresponding eight-bit sensor reading in binary is 00000001; when the line is further to the left, the corresponding reading in binary is 10000000. Due to the width of the tape, when the line is centered under the robot chassis (i.e., the error is zero), then the reading in binary 00011000. The eight-bit reading is therefore used to control the motion of the robot by applying a turning motion to zero the deviation of the line under the sensor.

A differential drive robot turns by commanding different speeds on the two driven wheels. Higher speed on the right wheel causes the robot to turn to the left and vice versa. Traditionally, to fully specify the movement of a differential drive robot, two parameters are provided, the forward speed V_F (also referred to as common-mode speed) which is applied to both wheels and the steering effort V_S. Positive values of V_S cause the robot to turn to the

left whereas negative values of V_S cause the robot to turn to the right. Here we assume that no clipping occurs, i.e., -100 <= $V_F + V_S/2$ <= 100.

The speed guidance to each wheel is then given by the following two equations.

$$V_L = V_F - \frac{V_S}{2} \quad (3)$$

$$V_R = V_F + \frac{V_S}{2} \quad (4)$$

Both algorithms generate desired PWM effort for the left and right wheel respectively by taking into account a pre-specified forward speed while dynamically calculating the steering effort in order to minimize the deviation of the line under the sensor.

Algorithm Description

The line following algorithm utilizes equations (3) and (4) to generate guidance. There are three distinct steering levels which depend on the three discrete levels of lateral error conveyed by the downward looking IR sensor. We refer to the corresponding steering efforts as "high", "med" and "low". This is the simplest approach to a line following behavior. The code first checks for the situation when the line is centered, in which case it commands no steering. It then checks for the sign of the error and then checks for the amplitude of the error. A binary value of 00000001 or 00000010 or 00000100 indicates the line is to the right hence the robot should turn to the right, and vice versa. The magnitude of the error is then determined. This is done by determining if the line is as far away as can be detected, i.e. the IR sensor reads 10000000 or 00000001. If so, the function controlling the wheels' speed is called with the right wheel given the main speed added by the big turn parameter multiplied by t, x + t*high, and the left wheel given the main speed minus the big turn parameter multiplied by t, x – t*high. If that is not the case, the program checks if the IR sensor's bits corresponding to the medium turn are 1, and passes the same parameters as for the big turn, changing the turn parameter from high to med. Last, if the IR sensor doesn't correspond to a big or medium turn the program checks if it corresponds to a small turn, i.e. if the third bits from left and right are 1, and passes the function the same parameters, changing the turn parameter from med to low. The program keeps cycling though this loop until we manually stop the robot or simulation. The code segment is shown in Figure 5.

```
if (IR == 0x18) {
    moveEffort(s,s);
}
else {
    if (IR & 0x01 || IR & 0x02 || IR & 0x04) {
        t = 1;
    else
        t = -1;
    if ( IR & 0x01 || IR & 0x80 )
        moveEffort(s + t*high, s - t*high);
    else if ( IR & 0x02 || IR & 0x40 )
        moveEffort(s + t*med, s - t*med);
    else if ( IR & 0x04 || IR & 0x20 )
        moveEffort(s + t*low, s - t*low);
}
```

Figure 5 - Line Following Code.

RESULTS

To obtain comparative results between the physical and simulated experiments, the code shown in Figure 5 was compiled and loaded on the robot as well as the simulator. On the physical experiment, the robot's location was obtained by the mocap system whereas in the simulator a precise location was collected by peeking into the robot's simulated location. Note that in a typical program run on the simulator, this information is not available but was made available in this version to facilitate data collection.

The case analyzed here utilizes a common mode speed of 20% effort, and differential speeds of 6%, 10% and 20% for the three levels of steering corresponding to the three levels of error returned by the downward looking sensor. Figure 6 depicts the physical and simulated trajectory. The two stars show the location of the robot at the start (red) and 0.4 meters into the run (lighter color). The robot is initially traveling towards higher Y, effectively traversing the course in a counter-clockwise direction. As expected, the trajectory is largely similar due to the self-correcting nature of the line following algorithm. The actual trajectory appears to have higher frequency components

suggesting higher levels of oscillation, and in certain cases (i.e., at the turn near (-0.4, -1)), there is a non-trivial difference between the physical and simulated robot's trajectory. In order to obtain additional insight, the lateral error was calculated both for the physical and simulated robot. The lateral error is calculated by taking each (x, y) sample of the robot's location and determining the shortest distance perpendicular to the tangent of the course.

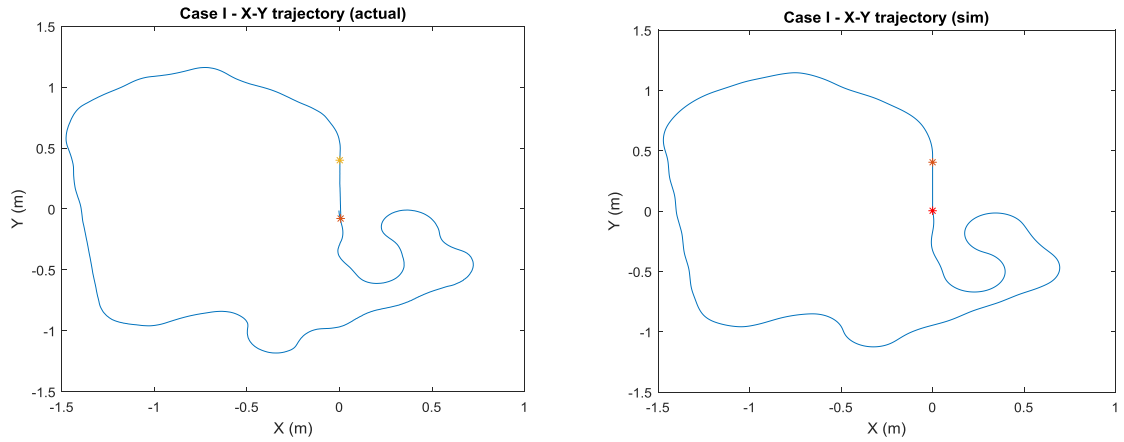

Figure 6 – Case I – Trajectory comparison.

Figure 7 depicts the lateral deviation comparison. Because the data collection rate was different on the simulator versus the physical robot, data was interpolated and then plotted versus the normalized course position. This allows us to better compare the data since the x-axis location corresponds to a specific position on the course, something that would not be the case if data was plotted versus time. Several things become apparent by looking at Figure 7. First, the overall magnitude of the deviation is very similar and for most part of the course ranges at below 2 cm. However, peak deviations appear at different locations. The actual robot has the highest peak at position 0.6 whereas the simulated robot has the highest peaks towards the end of the course. This implies that the response of the motors is different between the real and simulated robot, something that appears to affect the robot's ability to track the line while on turns of different curvature. The real robot has difficulty tracking the turn near (-0.4, -1.1) whereas the simulated robot seems to have difficulty tracking the two constant radius turns at locations (0.4, 0) and (-.3, -0.5). This suggests that the motor model suffers from inadequate fidelity, something that propagates into the ability of each version to track course curves.

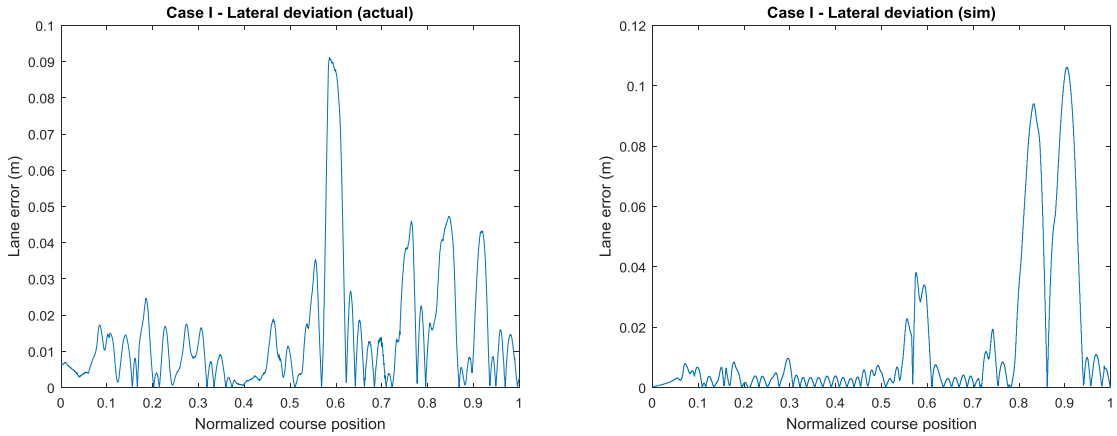

Figure 7- Case I – Lateral Deviation Comparison.

Figure 8 depicts the comparison between the heading of the physical and simulated robot, again plotted against the normalized course position. As expected, the overall trend matches but there are small differences in the number of oscillations, especially during traversal of straight segments. For example, for the portion within the normalized position range of [0.4, 0.47], the actual robot maintains a straight heading whereas the simulated robot is keeping a small but non-decaying oscillation. Again, this suggests that the fidelity of the motor model does not adequately

capture the response of the physical motors. The presence of oscillations suggests that the motor model requires additional damping or reduced time constant response.

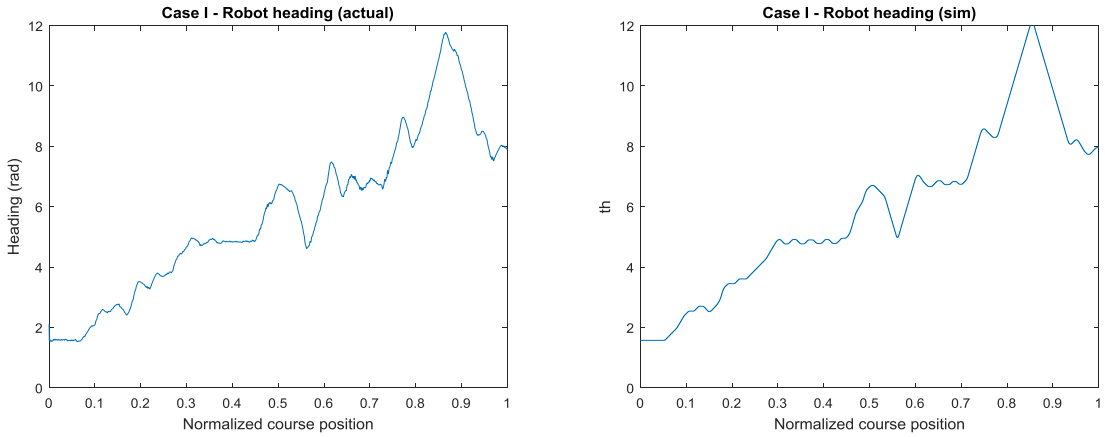

Figure 8 – Case I - Robot Heading Comparison.

A final comparison for Case I is shown in Figure 9. Several things are obvious when looking at this comparison. First, the baseline speed varies by almost 20% between the physical robot and simulation. Recall that in both cases, the common mode speed is at 20% effort. In the physical robot, this yields approximately 0.45 m/sec velocity whereas in the simulated robot, it yields 0.36 m/Sec, an almost 25% difference. In addition, the frequency content of the velocity is much higher in the physical robot than the simulated robot.

The difference in average speed is a simple artifact of the model. As discussed earlier, the motor model was developed by using a 3rd order system identification process using Matlab. The data utilized for the system identification spread the entire range of efforts (15% to 100%). During system identification, one of the estimations focuses on the scaling between input and output. Because the motors have a non-linear response, a lower order model can only match the input/output scaling on the average. In this case, lower effort level generates significant error between the real and simulated robot.

The difference in the frequency between the real and simulated robot supports the earlier suggestion that the simulated model is underdamped and has a much faster response than the real robot. This faster response allows the simulated robot to maintain a common mode speed that is constant, and only varies slightly during turn transients. This makes theoretical sense because when looking at equations (3) and (4) it is clear that the chassis speed given by $(V_L + V_R)/2$ should be constant, much like the data shown in the simulated robot case. However, increased damping causes individual wheel speeds to lag the command generating different velocities in the case of the physical robot.

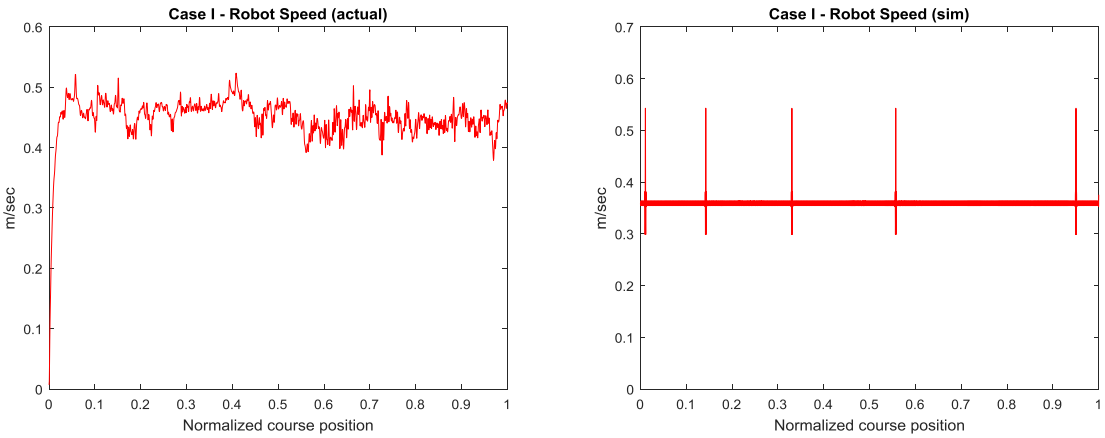

Figure 9 – Case I – Robot Velocity Comparison.

This is a typical case where the simulation model acts as in theory, however the physical model demonstrates all the aspects that were not included in the simulation.

CONCLUSIONS AND FUTURE WORK

Line following robots are very popular but have not been studied in detail, especially in terms of the ability of a simulated robot being used for testing in lieu of a physical prototype. This paper presented preliminary results of comparing the physical and simulated behavior of a line following robot. Results were mixed. The motor model is only an approximation and contains errors both in the steady state output as well as the response time. This is an artifact of the approach used to develop the model (automated system identification) as well as the system order used for modeling (3^{rd}), as well as the fact that the actual behavior of the motors is non-linear. The resultant system shows very similar trends between physical implementation and simulation however specific measures vary significantly. At the same time, if one evaluates the overall algorithm behavior in terms of the ability of the robot to follow the line, results are surprisingly robust. This is despite the differences between the model and the physical system. This is explained by the fact that the line following algorithm is a closed-loop algorithm so any errors in the response of the system are corrected via the feedback of tracking the line.

Ultimately, results suggest that for purposes of testing basic code functionality and general implementation approaches, simulation even with a lower fidelity model is adequate for testing. However, pushing the algorithm to the limit would expose the model deficiencies as marginal performance differs.

Clearly, future work involves addressing the deficiency of the model. A higher order model can be used, or a non-linear system could be developed to capture the nuances of the motor's response. An alternative approach would be to use the well-known technique of modeling non-linear systems by developing several linear models and switching at run time depending on the operational envelope. A repeat validation would then need to be performed by including boundary cases as a means to assess the new models' fidelity.

REFERENCES

Almasri, M.M., Alajlan, A.M., Elleithy, K.M. (2016). Trajectory Planning and Collision Avoidance Algorithm for Mobile Robotics Systems, *IEEE SENSORS JOURNAL, Vol. 16,* 5021-5028. doi:10.1109/JSEn.2016.2553126.

Barayyan, T.O., Barnawi, E.M. (2015). IR coding design and implementation for autonomous in-door mobile robot localization, *2015 5th National Symposium on Information Technology: Towards New Smart World (NSITNSW).* 1-5. doi: 10.1109/NSITNSW.2015.7176424.

Dupuis, J., Parizeau, M. (2006). Evolving a Vision-Based Line-Following Robot Controller. *The 3rd Canadian Conference on Computer and Robot Vision (CRV'06),*75-62. doi:10.1109/CRV.2006.32.

Hasan, K.M., Nahid, A. A. (2012). Implementation Of Autonomous Line Follower Robot. *2012 International Conference on Informatics, Electronics & Vision (ICIEV),* 865-869. doi:10.1109/ICIEV.2012.6317486.

Su, J.H., Lee, C.S., Huang, H.H., Chuang, S.H., Lin, Y.L. (2010). An intelligent line-following robot project for introductory robot courses, *World Transactions on Engineering and Technology Education, Vol.8,* 455-461.