ECE 484: Principles of Safe Autonomy	Spring 2022
Written by: ECE484 team	*HW0,MP0: Simple safety*
Website 	*Due Date: 4/2/22*

1 Introduction

Welcome! In this first assignment, you get to practice an invariance proof, play with the Gazebo simulator, and explore the gaps between simulations and analysis.

In part one (Section 2), you will use the inductive method to prove safety of the Automatic Emergency Braking scenario. You will need to show that, under some assumptions, the system is indeed safe, i.e., the vehicle will never hit the pedestrian. In part two (Section 3), you will work with a vehicle model in Gazebo and compare the simulations with the invariant and the assumptions in part one.

Please note that the written portion of this MP must be done individually and the programming portion of this MP can be done in groups of 2-4 people. More details for submission are given in Section 4. All the regulations for academic integrity and plagiarism spelled out in the student code apply.

Learning objectives

- Know and use inductive method for proving invariants
- Get introduced to ROS and Gazebo
- Understand gaps between simulation model and mathematical analysis model

System requirements

- Ubuntu 16.04
- ROS Kinetic
- Gazebo 9
- ros-kinetic-ackermann-msgs

2 Safety Analysis of Automatic Emergency Braking

We define an explicit model of scenario involving a vehicle and a pedestrian as shown in Figure 1. In this model, x_1, v_1, x_2, and v_2 are state variables. x_1 and v_1 correspond to the position and velocity of our vehicle. x_2 and v_2 correspond to the position and velocity of the pedestrian. We use the convention that $d(t)$ is the *valuation* of the state variable at time t. That is, $d(0) = x_2(0) - x_1(0) = x_{20} - x_{10}$, $d(1)$ is the value of d after the program is executed once, $d(2)$ after the program is executed a second time, and so on. Similarly, we can refer to other state variables in the same manner e.g. $x_1(t)$ and $x_1(t+1)$ refer to the valuation of x_1 in two different time instances. D_{sense} is a constant, which is the sensing distance: if $d(t) \leq D_{sense}$, the vehicle applies the brakes and decelerates.

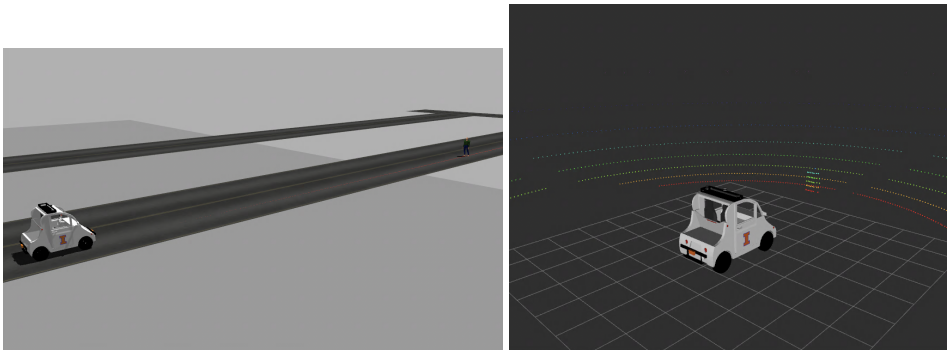

Figure 1: Vehicle and pedestrian on a single lane. *Left:* Initial positions of the two are at x_{10} and x_{20}, respectively. *Right:* Vehicle undergoes constant deceleration a_b once the pedestrian is detected by the vehicle.

SimpleCar$(D_{sense}, v_0, x_{10}, x_{20}, a_b)$, $x_{20} > x_{10}$
Initially : $x_1(0) = x_{10}$, $x_2(0) = x_{20}$, $v_1(0) = v_0$, $v_2(0) = 0$
$s(0) = 0$, $timer(0) = 0$, $timer2(0) = 0$
$d(t) = x_2(t) - x_1(t)$
if $d(t) \leq D_{sense}$
    $s(t+1) = 1$
    if $v_1(t) \geq a_b$
        $v_1(t+1) = v_1(t) - a_b$
        $timer(t+1) = timer(t) + 1$
        $timer2(t+1) = timer2(t)$
    else
        $v_1(t+1) = 0$
        $timer(t+1) = timer(t)$
        $timer2(t+1) = timer2(t)$
else
    $s(t+1) = 0$
    $v_1(t+1) = v_1(t)$
    $timer(t+1) = timer(t)$
    $timer2(t+1) = timer2(t) + 1$
$x_1(t+1) = x_1(t) + v_1(t)$

We will use the above model and we call it model $\mathcal{A}$. Consider the following invariant:

Invariant 1. Over all executions of $\mathcal{A}$, $timer(t) + v_1(t)/a_b \leq v_0/a_b$.

Problem 1 (10 points)**.** Is $0 \leq v(t) \leq v_0$ an invariant of $\mathcal{A}$? No need to write a complete proof; a two sentence argument would suffice.

Problem 2 (10 points)**.** Is $timer(t) \leq v_0/a_b$ an invariant of $\mathcal{A}$? Explain why. Can we use the induction method to prove this invariant? If so, present your proof.

Hint: You may find the usage of other invariants handy in your proof.

Problem 3 (15 points)**.** Let us now introduce some delay in the sensing-computation-actuation pipeline, say T_{react}. This could model cognitive delay of a human driver or processing delay in electronics and computers. Assume we have exactly T_{react} seconds delay between the sensing of the pedestrian and the application of the brakes (the **start** of the deceleration). Moreover, let us also introduce acceleration to the vehicle: whenever the vehicle is outside the sensing distance, the vehicle undergoes constant acceleration a_s. Rewrite the new updated model with the sensing delay and vehicle acceleration.

Problem 4 (5 bonus points)**.** Identify additional assumptions on $x_{20}, x_{10}, D_{sense}$ under which the system is safe. That is, come up with an new invariant such that you can prove this invariant using the method of induction, the assumptions, and the previously proved invariants. (Note: This question is much more difficult than previous one. If you get stuck with it, try to solve coding section first then get back to this one)

Hint: Try the assumptions: $x_{20} - x_{10} \geq D_{sense}$ and $D_{sense} > v_0^2/a_b + 2v_0$ and the invariant $d(t) > 0$.

3 Testing Automatic Emergency Braking with Simulations

In this second part, you will run the vehicle and pedestrian scenario in Gazebo simulator to cross-validate your analysis and assumptions. Note that the simulated vehicle dynamics is based on but slightly different from the SimpleCar model discussed above. One major difference is that for the SimpleCar model, the time step is 1s but for the simulated vehicle, the time step is 0.01s. In later MPs, you will have the chance to play around with more detailed vehicle models.

The vehicle starts from position $(0, 0)$ and cruises down a straight road in the x-direction. A static pedestrian is placed at position $(60, 0)$ in the middle of the road. The vehicle uses LIDAR to detect the pedestrian and once the pedestrian is detected ($D_{sense} \leq d(t)$), the vehicle will start braking at constant deceleration user provided. The vehicle will keep decelerating until its speed reaches 0.

You will try different values of the parameters D_{sense}, v_0, a_b, T_{react} and record corresponding $d(t)$ after the vehicle is stopped. You will need to compare the result from simulation with the invariants you proved in previous part.

3.1 Documentation of Provided Files

The supporting code is available from this git repo. The provided code for MP0 is located in `./src/mp0/src` folder. In this assignment, you need not implement anything because we have given you a nifty little function that you have to run with different parameters. However, we strongly encourage you to read through the code. You will learn ROS mechanics from this and you will have to write modules in later MPs and for your project. The important files are:

controller.py This file contains the vehicle model and the controller to determine the movement of the vehicle in the Gazebo simulator. The vehicle will run straight at constant speed at while the pedestrian is not detected. After the pedestrian is in the sensing distance from the vehicle, the vehicle will brake with constant deceleration provided by the user. The calculated velocity will be sent to the vehicle model to determine the next state of the vehicle. The result will be sent to Gazebo simulator through ROS topic. The controller is running at 100Hz, i.e. each simulation step is 0.01s.

lidarProcessing.py This file contains code to process LIDAR data. It will take raw point cloud data from the simulated sensor and transform it into a *bird's eye view*, which is then published through a ROS topic, and can be further processed by other parts of the simulation. Detailed information about point cloud data type and algorithm used to process data can be found here.

positionDetector.py This file contains code that uses the bird's eye view constructed from `lidarProcessing.py` and detects the position of the pedestrian in the bird view image. The position of the pedestrian in the bird eye view is then published through another ROS topic and can be further used by other parts of the system. Detailed information about algorithm used can be found here.

safetyDector.py This file contains code that use the pedestrian position calculated from positionDetector and compute the distance between LIDAR and the pedestrian. The distance calculated is compared with sensing distance to determine if the vehicle should start braking.

main.py This file contain the main function of the MP. You can run the file with four parameters, `--`d_sense is the sensing distance of the LIDAR D_{sense}, `--`v_0 is the initial velocity of the vehicle v_0, `--`a_b is the deceleration rate a_b and `--`t_react is cognitive delay for human driver T_{react}. Note that `--`d_sense should not exceed 20. Therefore, you can run the vehicle using command

```
python main.py --d_sense 15 --v_0 5 --a_b 5 --t_react 0.00
```

set_pos.py This is a utility function that allows you to set position of the vehicle model without restarting the simulator. You can set the position of the vehicle using command

```
python set_pos.py --x 0 --y 0
```

3.2 Short Gazebo Tutorial

The code for MP0 will come together inside the ROS workspace, which will also be used in later MPs. In this MP, you will work with vehicle models in Gazebo simulator using ROS. You should try to set up a virtual machine with required software environment by following the guideline provided on the course website. To run the simulator, you should first go to the root directory of the ROS workspace you downloaded from git repository where you should see a src folder. The next step is to run command

```
catkin_make
```

in the folder. There should be no error during the execution of the command and when finished, you should see two folders devel and build been generated.
The next step is to run command

```
source ./devel/setup.bash
```

in the root directory of the ROS workspace you downloaded. This command should be executed every time before you try to run the simulator from a new terminal.
After all the previous setup steps are finished, you can start the simulator by running command

```
roslaunch mp0 mp0.launch
```

You should be able to see two windows opened as shown in Figure 2 and 3.

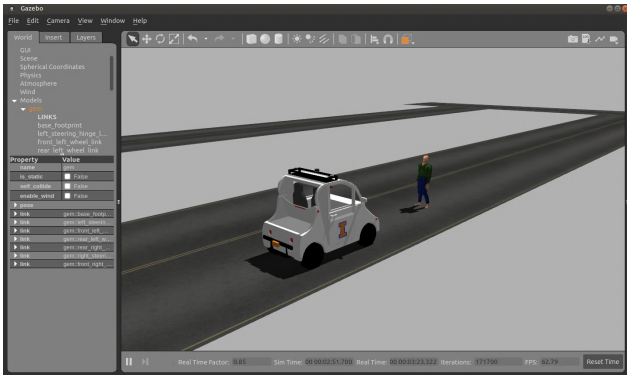

Figure 2: The Gazebo window. The vehicle is stopped before the pedestrian.

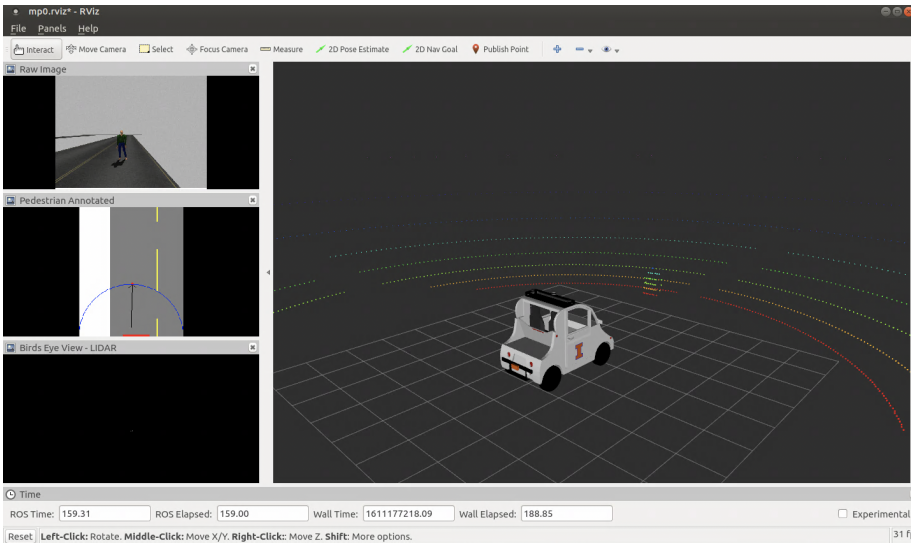

Figure 3: The RIVZ window corresponds to the Gazebo window in Figure 2.

Gazebo Window The window shown in Figure 2 is the Gazebo simulator window. You should be able to see the vehicle and pedestrian in the simulator window.

Rviz Window The window shown in Figure 3 is the Rviz window. Rviz is a ROS graphical interface that allows you to visualize ROS topics. In this MP, we will mainly use Rviz to visualize LIDAR data from the vehicle. In this MP, the Rviz window is divided into 4 different sections.

The window on the right will show a visualization of the point cloud data from the LIDAR. You can see that the has detected the pedestrian in front of the vehicle.

The window on the top left (Raw Image) is showing the video streaming from the camera on the vehicle.

The window on the middle left (Birds Eye View - LIDAR) will show the raw bird view from the LIDAR data. You can see a small white spot in the window as the detected pedestrian.

The window on the bottom left (Pedestrian Annotated) will show the relative position between the vehicle and the detected pedestrian. The detected position of the pedestrian will be marked as red spot in the

window with an arrow connecting the LIDAR and the detected pedestrian.

The Birds Eye View - LIDAR and Pedestrian Annotated windows will show "No Image" when Rviz is opened since they are displaying topics published from MP0 code.

3.3 Simulation Problems: Zero Reaction Delay

Assume there is no reaction time. Therefore, the --t_react parameter will be set to default value 0 for all the problems in this section. For this case, you can try different combinations of values for parameter --d_sense, --v_0 and --a_b and observe final values of d(t) after the vehicle stopped.

Problem 5 (15 points)**.** Record a video for one example execution of this scenario. The video should include the Gazebo window, the Rviz window, and the terminal window that runs MP0 code. Provide a link to the video and include the link in the report. In addition, plot the time versus state variables ($x_1(t)$, $x_2(t)$) for this execution.

Problem 6 (20 points)**.** Fix d_sense and a_b, and choose 5 different values for v_0. Try to choose interesting values or "corner cases" that could make the system safe and unsafe.

(a) For each of these choices, plot the final value of d(t) vs your choice of v_0.

(b) Are there any choices that violate the safety of the system?

(c) Try reducing d_sense. How does this affect the safety of the system in relation to v_0?

3.4 Adding Reaction Delay

In this case, we will assume that there is an exact reaction time for the vehicle to start braking after the vehicle detects the pedestrian as mentioned in 4. You can choose the amount of reaction time by setting the --t_react parameter. For this case, you can try different combinations of values for parameter --d_sense, --v_0, --a_b and --t_react and observe final values of d(t) after the vehicle stopped.

Problem 7 (15 points)**.** Record a video for one example execution of this scenario. The video should include the Gazebo window, the Rviz window, and the terminal window that runs MP0 code. Provide a link to the video and include it in the report. In addition, plot the time versus state variables ($x_1(t)$, $x_2(t)$) for this execution.

Problem 8 (15 points)**.** Fix d_sense, v_0, a_b, and choose 5 different t_react. Try to choose interesting values or "corner cases" that could make the system safe and unsafe.

(a) For each of these choices, plot the final value of d(t) vs your choice of v_0.

(b) Are there any choices that violate the safety of the system?

(c) Try modifying a_b. How do different choices of a_b affect the safety of the system in relation to t_react?

4 Report and Submission

For this MP, problems 1-4 must be done individually (Homework 0). Write solutions to each problem in a file named `<netid>_ECE484_HW0.pdf` and upload the document in Canvas. Include your name and

netid in the pdf. This should be individual work and you should follow the student code of conduct. You may discuss solutions with others, but not use written notes from those discussions to write your answers. If you use/read any material outside of those provided for this class to help grapple with the problem, you should cite them explicitly.

Problems 5-8 can be done in groups of 2-4. Each group should write a report that contains the solutions, plots, and discussions for all the problems. Your group should submit a single report with name `MP0_<groupname>.pdf` to Canvas. Please include the names and netids of all the group members and cite any external resources you may have used in your solutions.