Lab 6: ethics and autonomous vehicles

COMP130  
Instructor: John MacCormick

# Introduction

Driverless cars are a reality, but there are still unsolved problems related to the ethics of driverless car algorithms. In this lab we explore simplified scenarios to assess some of these ethical problems.

We also use simplified versions of some famous philosophical ideas to help our analysis. In this lab, our two ethical principles are:

* Utilitarianism: Choose the action which results in the greatest happiness or well-being for the greatest number of people
* Do-no-harm: Never take an action that will cause harm or suffering.

Utilitarianism is closely associated with the 18th-century British philosopher Jeremy Bentham. Do-no-harm is often used as a basic principle in bioethics, and also has connections to the 18th-century German philosopher Immanuel Kant. As optional preparation for the lab, you may wish to read the following 2016 New York Times article by John Markoff, or watch the video embedded in the article: [Should Your Driverless Car Hit a Pedestrian to Save Your Life?](https://www.nytimes.com/2016/06/24/technology/should-your-driverless-car-hit-a-pedestrian-to-save-your-life.html).

# Required preparation before the lab begins

Moral Machine, at <https://www.moralmachine.net/>, is a platform for experimenting with human attitudes to certain ethical problems. It was initially created as part of a research program by Jean-François Bonnefon, Azim Shariff and Iyad Rahwan, described in their 2016 Science journal article “The social dilemma of autonomous vehicles.” Before the start of the lab, each student must judge a set of scenarios at Moral Machine, make a note of the results, and answer the preliminary questions below. Save your answers in a document and bring them with you to the start of the lab session.

**Qu 1.** Go to Moral Machine at the link above. Optionally, read any background information available at this site. When ready, click on “Judge”, one of the main options at the top. This will lead to a set of scenarios. Read each scenario carefully, including the description which is available using the “show description” button. At the end, you will be presented with a set of results comparing your own ethical judgments to the average results of other participants. Write 2-3 sentences summarizing the differences and similarities in your judgments compared to the averages of others.

**Qu 2.** Consider the two ethical principles described above: utilitarianism and do-no-harm. Write 2-3 sentences explaining how your own Moral Machine results reflect these principles. Try to address the following questions. Which principle did you rely on more strongly? Did you employ both principles, only one, or neither?

This is the end of the required preparation before the lab begins.

# Beginning work with your partner

**Qu 1.** Both partners should paste their answers to question 1 above into the responses document.

**Qu 2.** Both partners should paste their answers to question 2 above into the responses document.

Now you are ready to work as a team on the rest of the lab.

# Modeling Scenarios

It is one thing for individuals to make judgments with the Moral Machine. It is another to program those decisions into vehicles that will affect large numbers of people. Eventually, we will be using simulations to explore the impact of encoding different ethical values into driverless cars. In order to do that, we will need to model the scenarios that a driverless car might encounter. To do so, this lab provides a module named autonomous\_vehicles.py. Download this file and place it in the same folder with any other files you are using for this lab, including a new file called lab06.py.

The autonomous\_vehicles module defines a class of objects named Scenario. A Scenario object encapsulates all the information about the situations that our simulated self-driving cars will encounter. In particular, it contains:

* the number of passengers in the car
* the state of the left and right lanes, including:
  + whether a barrier exists in the lane
  + the number of pedestrians that will be killed if the car is in that lane
  + whether the pedestrian crossing signal for that lane is on or off

For the purposes of our simulations, we will identify the lanes as *left* or *right* from the observer's perspective. That is, when you are looking at one of the Moral Machine pictures, the left lane is the one on *your* left.

The example code below shows how to import the autonomous\_vehicles module, create a Scenario object, and then output the result of calling each of the methods that the object supports.

import autonomous\_vehicles as av

s = av.Scenario(3, True, 0, False, False, 5, True)

print ("Passengers: " + str(s.num\_passengers()))

print ("Left Lane:")

print (" Barrier exits: " + str(s.left\_barrier\_exists()))

print (" Pedestrians: " + str(s.left\_num\_pedestrians()))

print (" Crossing signal on: " + str(s.left\_crossing\_signal\_on()))

print ("Right Lane:")

print (" Barrier exists: " + str(s.right\_barrier\_exists()))

print (" Pedestrians: " + str(s.right\_num\_pedestrians()))

print (" Crossing signal on: " + str(s.right\_crossing\_signal\_on()))

**Qu 3.** Encapsulate the example code above into a function called print\_basic\_scenario(). Hint: don’t put the import statement inside the function; this should go at the top of the file.

**Qu 4.** Write a function called example\_scenario() that constructs a scenario with a barrier in the left lane and two pedestrians in the right lane.

The Scenario class has a method named randomize(). This method randomizes the values in a Scenario object. The randomize() method also ensures that the the object is consistent. For example, if there is a barrier in the left lane, there will not be any pedestrians in the left lane. The randomize() method will be useful for our simulations below.

**Qu 5.** Write a function called random\_scenarios() that creates and prints 10 random scenarios.

**Qu 6.** (Answer in responses document) What are the largest and smallest number of pedestrians observed in any lane when you generate random scenarios as in the previous question? Do you ever observe a barrier in both lanes? Why or why not?

## A selfish car

**Qu 7.** Define a function with signature selfish\_car(s) that controls the car so that it preserves the passengers’ lives. The parameter s is a Scenario object. The function returns either the string 'Stay' or the string 'Swerve' indicating if the car should stay in its lane or swerve into the other lane. The function should assume that the car is in the left lane.

The code below contains a test that partially checks that your function is working correctly.

s = av.Scenario(3, False, 5, True, True, 0, True)

assert selfish\_car(s) == "Stay", "Swerved but should have stayed."

print("Success!")

**Qu 8.** Encapsulate the code above into a function called test\_selfish\_car(). Add a new test for another scenario that tests the Swerve behavior of your selfish car.

## A utilitarian car

Below you will be writing a function that makes decisions based on Bentham’s utilitarian principle. Because this principle will choose the option with the fewest fatalities, we will need to be able to compare the number of fatalities that occur when the car Stays and when it Swerves.

**Qu 9.** Write a function with the signature num\_dead(lane, s), where lane is 'Left' or 'Right' and s is a Scenario object. The function should return the number of fatalities that would occur in scenario s if the car is in the specified lane.

**Qu 10.** Write a function test\_num\_dead(). This function should test the function from the previous question. It must achieve statement coverage. Note: You may use the same Scenario in more than one assert.

**Qu 11.** Define a function with signature util\_car(s) that controls the car according to Bentham's utilitarian ethical principle. The parameter s is a Scenario object. The function returns either the string 'Stay' or the string 'Swerve' indicating if the car should stay in its lane or swerve into the other lane. The function should assume that the car is in the left lane.

**Qu 12.** Write a function test\_util\_car(). This function should test the function from the previous question. It must achieve statement coverage.

## Estimating the cost of selfishness

**Qu 13.** Write a function with signature simulate\_car(car\_type). The parameter car\_type is a string, either 'Selfish' or 'Util'. The function should simulate 10000 randomized scenarios using the given type of car, and print out the average number of fatalities per scenario.

**Qu 14.** (Answer in responses document.) Use the function from the previous question to run some experiments and estimate the average number of additional fatalities caused by a selfish car in a single scenario. Give your results and explain them in two or three sentences.

The remainder of the lab is optional.

## Optional: a Kantian car

According to some interpretations of Immanuel Kant’s philosophy, the car should never take an action that causes a fatality. We can also think of this as the do-no-harm approach. So the car can swerve to save lives if no one else will be killed. But if swerving would cause even one death, then the car must not swerve. Write a kant\_car(s) function implementing this functionality, and another function to test it. Extend your simulation function from earlier and run experiments with the Kantian car. Comment on the results in your responses document.

## Optional: visualize the results

Use the plot\_results() function given below to visualize how the number of fatalities varies as we increase the proportion of selfish cars. Paste the graph into your responses document and comment briefly on these results.

from matplotlib.pyplot import \*

def plot\_results():

x = []

y = []

s = av.Scenario()

pct\_selfish = 1

# set the starting percentage of selfish cars

for fraction in range(100):

total\_dead = 0 # for each fraction of selfish cars

for i in range(1000):

s.randomize() # 1000 trials per fraction.

if random.randint(1, 100) < pct\_selfish: # Pick selfish or utilitarian

if selfish\_car(s) == "Stay":

total\_dead = total\_dead + num\_dead("left", s)

else:

total\_dead = total\_dead + num\_dead("right", s)

else:

if util\_car(s) == "Stay":

total\_dead = total\_dead + num\_dead("left", s)

else:

total\_dead = total\_dead + num\_dead("right", s)

x.append(fraction / 100)

y.append(total\_dead / 1000)

pct\_selfish = pct\_selfish + 1

# Generate a plot of the x and y values.

scatter(x, y, ) # plot x and y.

title("Fatalities Due to Selfish Cars") # Title for the plot.

xlabel("Fraction of Selfish Cars") # Label the x axis.

ylabel("Deaths/Scenario") # Label the y axis

grid(True) # Turn on the grid lines.

show() # Make the graph appear.

**Acknowledgment.** This lab was originally authored by Grant Braught. It was adapted and edited by Lev Fruchter and John MacCormick.