CMSC 722, Spring 2018, Project 2:

Planning and Acting with Unreliable Steering

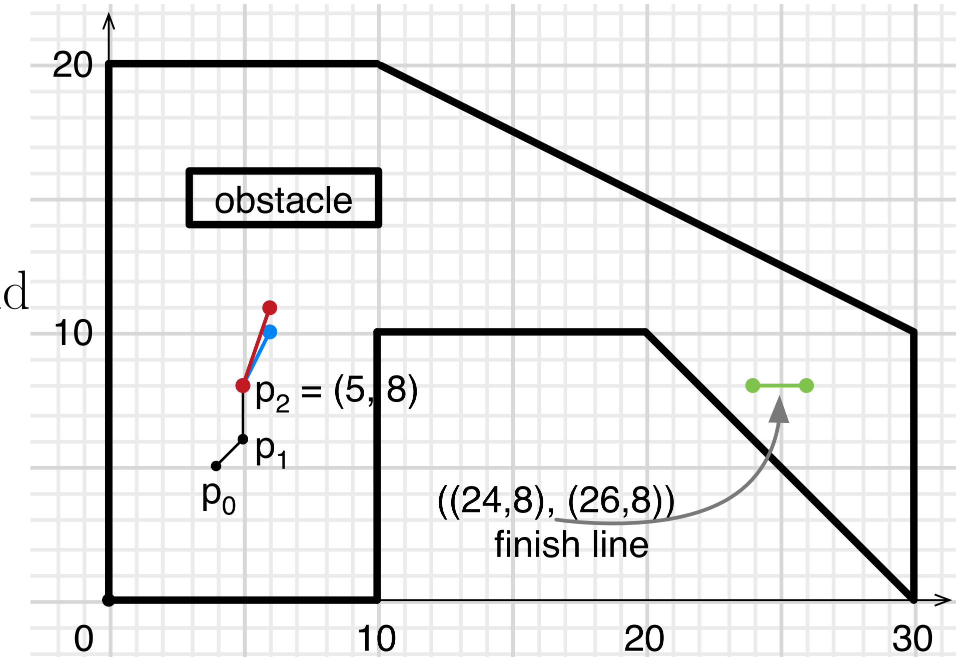
Last update April 24, 2018

- Due date: May 5, 11:59pm
- Late date (10% off): May 7, 11:59pm

Racetrack problem with unreliable steering

- Starting point, finish line, walls are the same as in Project 1
- Current state $s_{i-1} = (p_{i-1}, z_{i-1})$
  location $p_{i-1} = (x_{i-1}, y_{i-1})$ and
  velocity $z_{i-1} = (u_{i-1}, v_{i-1})$
- You choose new velocity $z_i = (u_i, v_i)$, where

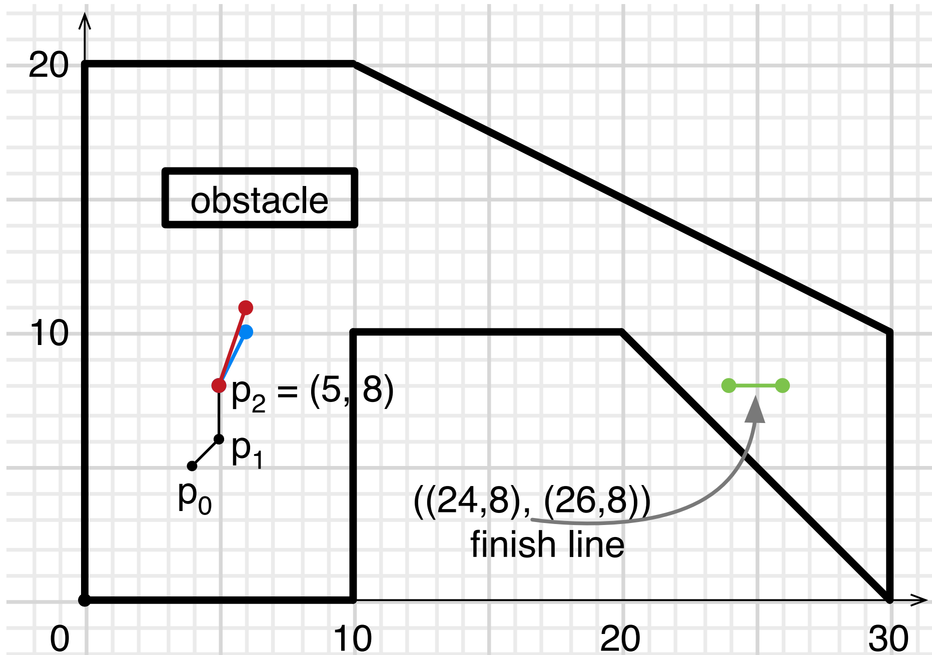
  $u_i \in \{u_{i-1} - 1,\ u_{i-1},\ u_{i-1} + 1\}$,
  $v_i \in \{v_{i-1} - 1,\ v_{i-1},\ v_{i-1} + 1\}$.



- Control system introduces random error $e_i = (q_i, r_i)$
  - $q_i, r_i \in \{-1, 0, 1\}$ are independent random variables (next slide)
- New location $p_i = p_{i-1} + z_i + e_i = (x_{i-1} + u_i + q_i,\ y_{i-1} + v_i + r_i)$
- New state $s_i = (p_i, z_i)$

Movement errors

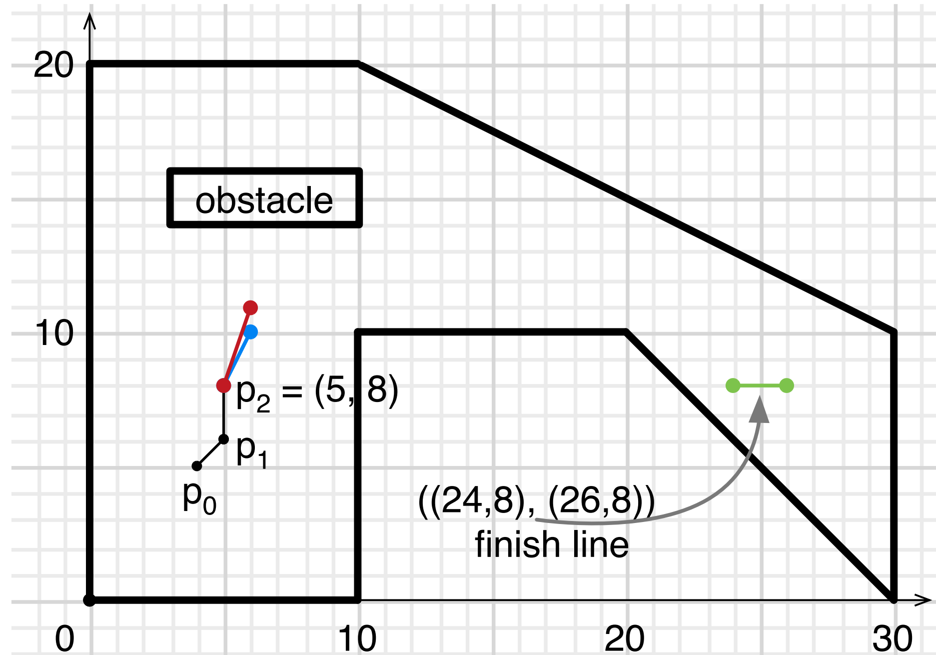
- Suppose you choose $z_i = (u_i, v_i)$
- Steering error $e_i = (q_i, r_i)$
  - $q_i, r_i \in \{-1, 0, 1\}$ are independent random variables



- If $|u_i| \leq 1$
  then $\Pr[q_i = 0] = 1$ and $\Pr[q_i = 1] = \Pr[q_i = -1] = 0$
  else $\Pr[q_i = 0] = 0.6$ and $\Pr[q_i = 1] = \Pr[q_i = -1] = 0.2$
- If $|v_i| \leq 1$
  then $\Pr[r_i = 0] = 1$ and $\Pr[r_i = 1] = \Pr[r_i = -1] = 0$
  else $\Pr[r_i = 0] = 0.6$ and $\Pr[r_i = 1] = \Pr[r_i = -1] = 0.2$

Example

- State $s_2 = ((5,8),(0,2))$
  $\qquad\qquad\qquad\qquad p_2, \quad z_2$
- You choose
  $z_3 = z_2 + (1,0) = (1,2)$
- Control error $e_3 = (0,1)$
- New location $p_3 = p_2 + z_3 + e_3$
  $= (5,8) + (1,2) + (0,1)$
  $= (6,11)$

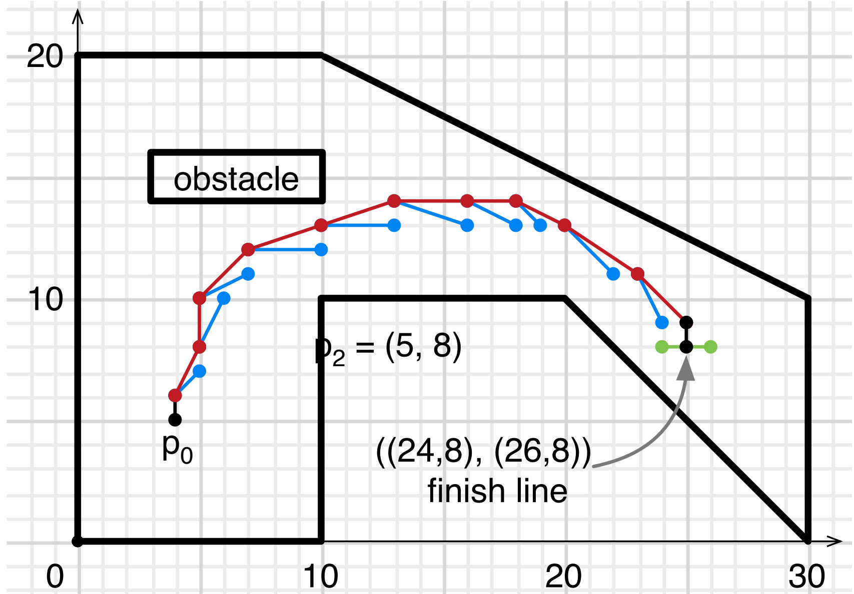


- New state $s_3 = (p_3, z_3) = ((6,11),(1,2))$

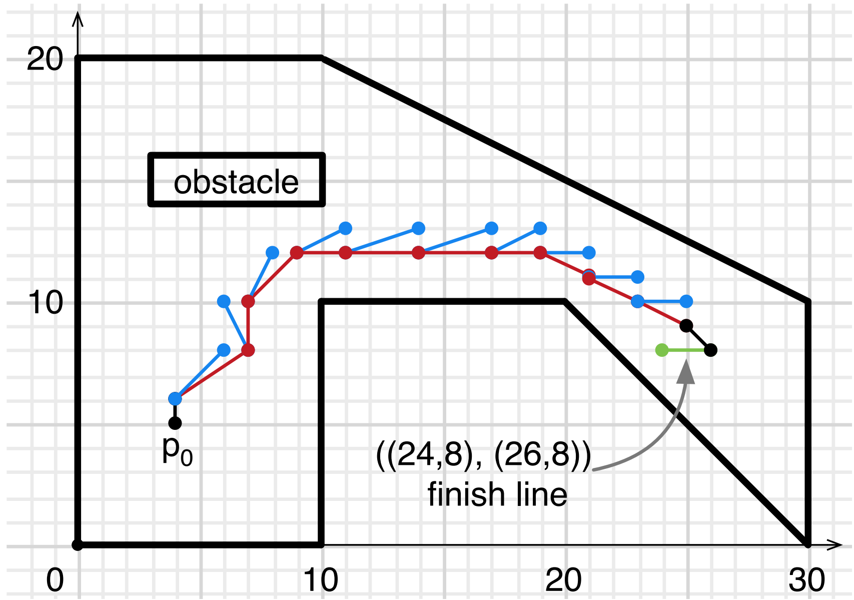
- The control error doesn't change velocity, just your position
  - Unrealistic, but it makes the problems easier to solve

Two Low-Probability Examples

- Vehicle always pulls to the left
  - has probability $(0.12)^{10}$

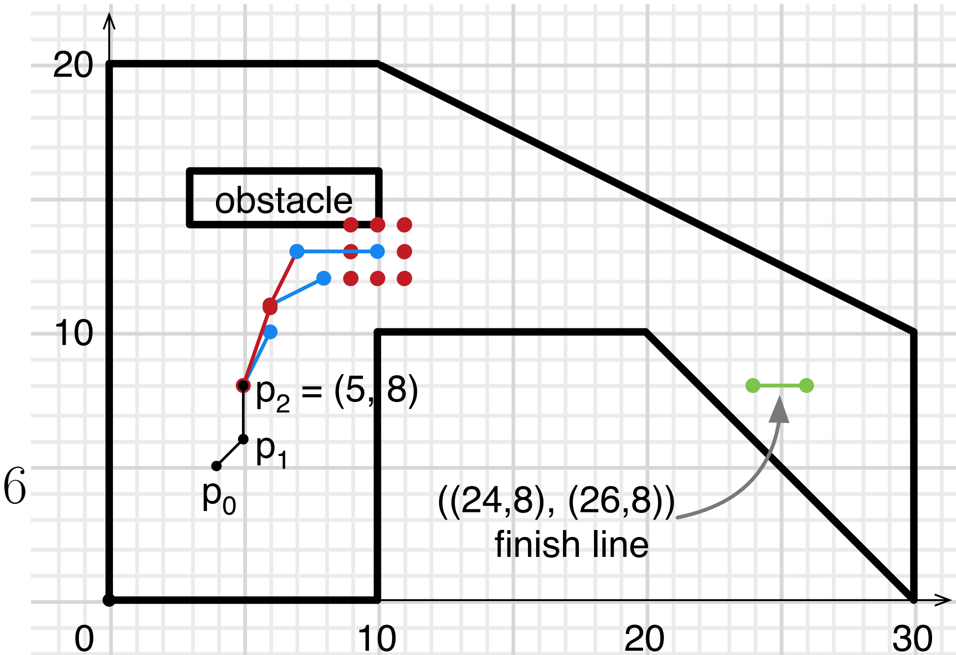


- Vehicle always pulls to the right:
  - again, probability $(0.12)^{10}$



Example

- State $s_4 = ((7, 13), (2, 1))$
  $$p_4, \quad z_4$$
- You choose
  $$z_5 = z_4 + (1, -1) = (3, 0)$$
- Crash if control error e_5 is $(1, 0)$ or $(1, -1)$
  - $\Pr = 0.2(0.6) + 0.2(0.2) = 0.16$



- Want to minimize the probability of crashing
  - Ideally 0, but that isn't always possible
- Also want to minimize the number of moves
  - Use a weighted combination
- Use one of the algorithms in Chapter 6
  - Possibly with modifications

Other comments

- You may use any of the code I gave you for Project 1, and any of the code your team developed for Project 1
  - You can modify it if you wish

- You'll need to write two SSP algorithms
- One or both might need a heuristic function
  - Don't use `h_ff1` or `h_ff2`, they'll take *way* too much time
  - You might want to use `h_esdist` or `h_walldist`

What you need to do

1. Write a Python function `proj2a.main`$(s, f, walls)$ to choose the next velocity
   - Use one of the algorithms in Chapter 6 (you can choose which one)
   - We'll give you a supervisor program
     - At each turn i, it will call `proj2a.main`$(s_{i-1}, f, walls)$
     - `proj2a.main` should search for better and better choices for $z_i = (u_i, v_i)$
       - e.g., additional iterations or additional Monte Carlo rollouts
     - Should print each choice, followed by a linebreak, to a file called `choices.txt`

```
(2, 2)
(1, 3)
(1, 2)
(1, 2)
```

     - ~~Shouldn't exit unless it has exhausted its search space~~ You may have it exit whenever you think is best

What you need to do

- The supervisor will do the following:
  (a) call `proj2a.initialize` if your `proj2a` file contains such a function
    - For `proj2a.initialize` to be useful, it will probably need to write to a file. Otherwise its work will be lost when the process exits.

  (b) call your program and let it run for an amount of time t_{search}
    - e.g., 500 or 1000 milliseconds

  (c) kill your program and use the last velocity it recommended

  (d) choose $e_i = (q_i, r_i)$ using the probability distribution discussed earlier

  (e) compute the new state, and check whether the run has finished
    - if the vehicle crashed, or reached the finish line with velocity $(0, 0)$

  (f) If the run hasn't finished, it will call your program again, with the new current state

What you need to do

2. Write another Python function, `proj2b.main`$(s, f, walls)$

- It should do the same kind of thing that `proj2a.main` does
- But use a different SSP algorithm
- One that you think will be interesting to compare with `proj2a.main`

- Note: the supervisor program has the name `proj2a` coded into it. To use it with `proj2b`, you'll need to change the code

What you need to do

2. Formulate some questions and/or hypotheses about how the algorithms will perform under various conditions

- Design and peform experiments to answer your questions and/or test your hypotheses
- Details are up to you

- Some things you could consider:
  - randomly generate problems as in Project 1
  - try the suite of problems I gave you for Project 1
  - run the algorithms with
    - different amounts of search time t_{search}
    - different relative weights on the things you want to minimize (number of moves, and probability of crashing)

- Each data point should be an average of at least 50 runs

What you need to do

3. Write a report giving the results of your experiments

- Explain the motivation
  - What questions and hypotheses did you want to investigate?
- Include plots and tables similar to the ones you did for Project 1
- For each plot or table, what can you can conclude from it and why?
- What are your overall conclusions?
  - Did the experimental results confirm or refute your hypotheses?

- Format:
  - US letter paper, single column, 1-inch margins on all sides
  - Font size at least 11pt
  - At most 4 pages

Grading

- Evaluation criteria:
  - 35% correctness: – whether your program works correctly, whether your submission follows the instructions
  - 15% programming style – see the following
    - Style guide: https://www.python.org/dev/peps/pep-0008/
    - Python essays: https://www.python.org/doc/essays/
  - 15% documentation
    - Docstrings at start of file and in each function; comments elsewhere
  - 35% on the report itself
    - Adequacy of your experiments, statistical significance, clarity of presentation, quality of conclusions