16.32 Final Report: Dynamic Programming for Car Kinematics

Keenan Albee
Massachusetts Intsitute of Technology
77 Massachusetts Ave
Cambridge, MA 20139
917-531-4411
albee@mit.edu

***Abstract*—A dynamic programming scheme was implemented in Python to solve the optimal car motion problem. The Reeds-Shepp Car model was used over a state space $X \subset x \times y \times \theta$, with inputs $U \subset v \times \phi$. A simple collision checking algorithm was incorporated into the dynamic programming solver to allow for the placement of arbitrary rectangular obstacles in the state space. The solver was run over multiple demonstrative scenarios, including parallel parking, navigation around an obstacle, and a 180 degree turn. The algorithm runs in $O(e^n)$, where n is the number of state space and input variables, in this case 5. Even taking advantage of a few heuristic tricks, this borders on the edge of feasibility due to the curse of dimensionality. Future work might involve more computationally efficient approximate solutions to the optimization problem, including RRTs.**

TABLE OF CONTENTS

1. BACKGROUND

[1]

Dynamic Programming

Dynamic programming provides a numerical tool for approximating optimization problems. Specifically, dynamic programming is the discrete-time approximation of the Hamilton-Jacobi-Bellman equation

$$-J_t^* = \min_{u \in U} \mathcal{H}$$

and in the limit provides the exact solution for the optimal cost-to-go, J^*. By discretizing input actions and state, one effectively creates a mesh of state variables over which the combination of discretized state variables may be applied. [2]

Algorithm

The dynamic programming implementation is based off of that given by

2. METHOD

The Reeds-Shepp Car

Collision Checking

procedure DYNAMIC PROGRAMMING(X, U)
  $r \leftarrow a \bmod b$
  while $r \neq 0$ **do** ▷ We have the answer if r is 0
    $a \leftarrow b$
    $b \leftarrow r$
    $r \leftarrow a \bmod b$
  end while
  return b ▷ The gcd is b
end procedure

Figure 1. Simple Collision Checking

Dynamic Programming

procedure DYNAMIC PROGRAMMING(X, U)
  $J^* \leftarrow h(x(t_f))$ ▷ For admissiable final position
  for k iterations **do** ▷ k mesh iterations
    for X states **do** ▷ discretized state
      for U inputs **do** ▷ discretized inputs
        $x_{k1} \leftarrow f(x_k, u_k)$
        $b \leftarrow r$
        $r \leftarrow a \bmod b$
      end for
    end for
  end for
  return b ▷ The gcd is b
end procedure

Figure 2. Computational Dynamic Programming

Implementation

procedure EUCLID(a, b) ▷ The g.c.d. of a and b
  $r \leftarrow a \bmod b$
  while $r \neq 0$ **do** ▷ We have the answer if r is 0
    $a \leftarrow b$
    $b \leftarrow r$
    $r \leftarrow a \bmod b$
  end while
  return b ▷ The gcd is b
end procedure

Figure 3. Interpolation Logic

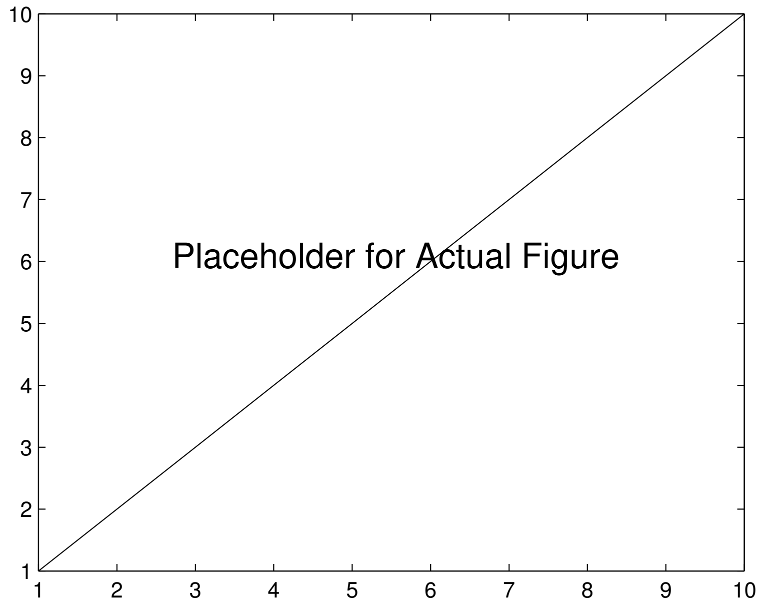

Figure 4. Here is an example of a figure that spans both columns.

3. RESULTS

Computational Complexity

Sample Scenarios

4. CONCLUSIONS

Discussion

Conclusion

APPENDICES

A. DYNAMIC PROGRAMMING ALGORITHM

[3]

ACKNOWLEDGMENTS

I would like to thank Professor Hall for teaching what was a challenging, but very interesting course.

REFERENCES

[1] S. LaValle, "Randomized Kinodynamic Planning," 2001.

[2] S. Hall, "Lecture 9: HJB Equation," pp. 1–32, 2018.

[3] Kirk, "Optimal control theory," pp. 1018–1018, 1971. [Online]. Available: http://doi.wiley.com/10.1002/aic.690170452

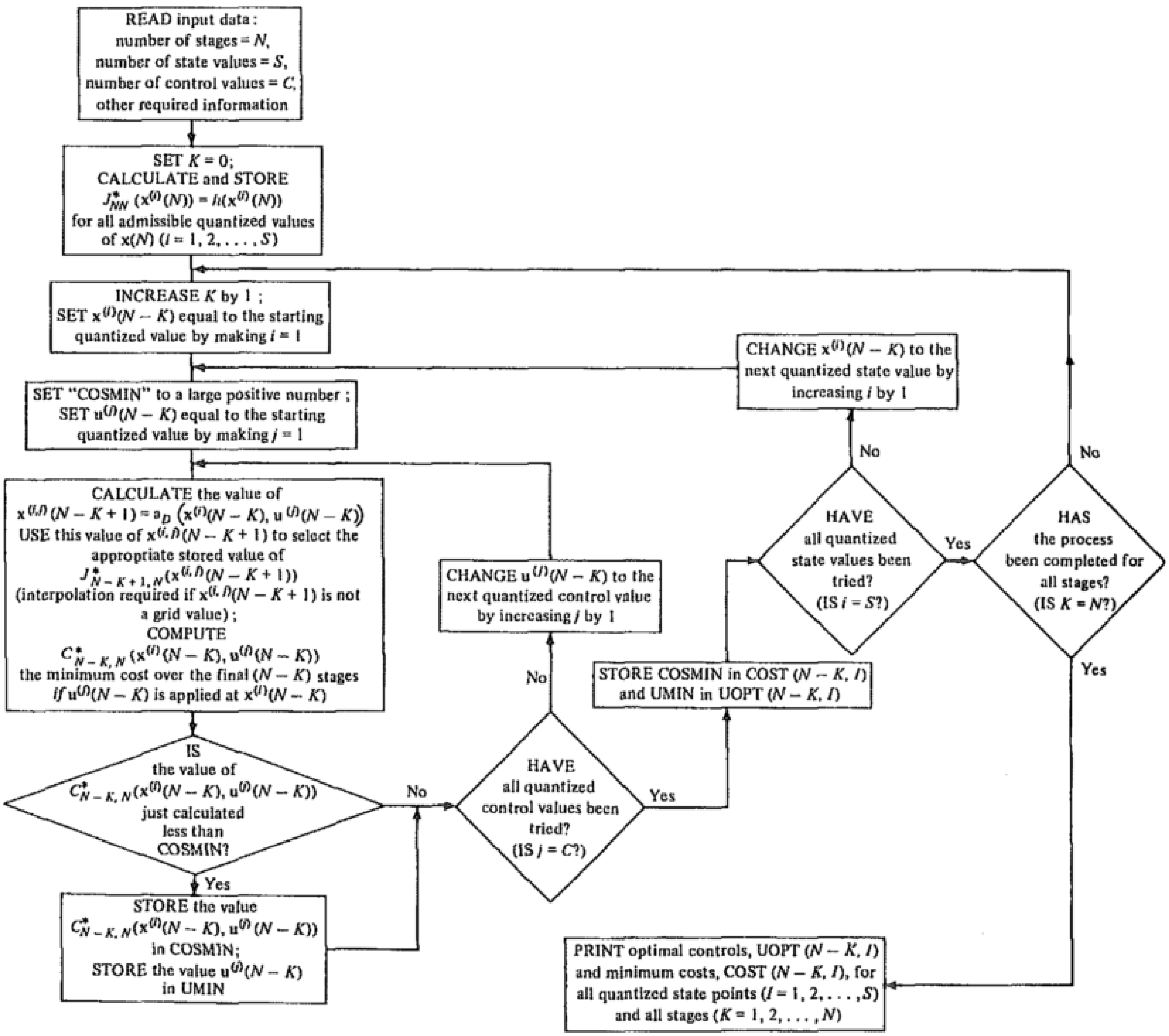

Figure 5. **The general dynamic programming algorithm.**