Application of Variable Offsets to Convex and Concave Polygons

Nicole Guymer and Niklas Delboi

March 2019

Abstract

A mathematical approach is presented showing a racetrack that is offset by different values along its shape. For this, different methods were tested to get the best solution that was accepted by the customer, the Oregon State University's Formula SAE Team. They want to generate an offset of a randomly selected racetrack in which the race car can drive autonomously and safely. Two different methods are presented and an evaluation of their implementation in this field study is given. For the selected solution, the track was split into points and the points were connected as a polygon. Then the midpoints of the polygon's edges were calculated and were offset with an individual value. Lastly, the offset midpoints were defined as the new vertices and a B-spline was generated through those points to create a smooth, continuous shape. This procedure was applied to both the inside and outside edge of the selected racetrack.

1 INTRODUCTION

The dream of a world filled with fully autonomous cars is fast approaching. While there are many different groups working on the technology, one group is taking a unique approach. Formula SAE is taking a step away from the consumer car and is instead focusing on autonomous race cars. Formula SAE is an international collegiate competition that challenges university students to design, build, and test a "small, formula-style vehicle" in order to compete against other university teams all over the world (1). One new subset of the competition is the Driverless Competition. In the Driverless Competition, the racetrack is not revealed to the teams ahead of time, which means that the car has to be able to learn the track as it drives. In order to do this, the car uses sensors to detect the edges of the racetrack, which are marked by yellow and blue cones, and builds a map of the track that can be stored on internal computers. Due to noise and the general uncertainty that is associated with sensors, the teams needs to keep the car a safe distance from the edges of the track while still allowing it enough freedom to drive efficiently. To do that our group decided to map a configuration space for the car that would adjust for the amount of uncertainty in the location of each cone around the track and offset the stored location for that cone by the corresponding uncertainty. This shrinks the width of the track significantly if the car is uncertain about the where exactly the edge of the track is but gives a much wider cross section in areas where it has high confidence. Our algorithm creates the configuration space for the race car by taking in the data from the vision sensors. We are able to map both the edges of the track and a configuration space in which the car can operate safely at all times.

2 BACKGROUND

In order to create the configuration space planner for this problem, we want to understand where the uncertainty in the cone locations comes from and how the car is determining that uncertainty. With this information, we are able to apply a few different mathematical techniques to map the edges of the track and the offset configuration space. The primary techniques used are polygonal offsets and B-splines.

2.1 Robotics

First, it is necessary to understand what a configuration space is. In robotics, the configuration space is defined as "a complete specification of the position of every point in the system" that the robot, or race car, can operate in (2). This simply means the map of the environment is reduced to only the area in which we expect the car to operate. This allows us to ignore the kinematics and dynamics of the car as these would typically be included in a state space, which is outside the scope of the class.

The need for a configuration space is driven by the existence of noise. Noise is sensor inaccuracies that can cause major problems when it comes time to interpret and act on data received from the sensors. Noise typically refers to the meaningless data that is collected during sensing and can come many sources. These sources include electrical noise from the batteries and wiring in the car or mechanical noise such as vibrations from the engine. These sources can saturate and lead to a misinterpretation of the data which can result in an inaccurate understanding of the world (3).

2.2 Polygons

Much of our solution relies on the application and manipulation of polygons to create the configuration space. Polygons can either be convex, no internal angle greater than 180°, or concave, and they must consist of only straight sides and must be a closed loop (4). Polygons can also either be simple or complex, meaning self intersecting. Our algorithm is able to handle any simple polygon.

2.3 B-Splines

To convert our data from a multi-sided polygon into a smooth, continuous track we fit B-splines to the points. B-splines ”use blending functions to form combinations of control points” which are used to approximate the spline that best fits the given data (5). B-splines are one of the more common methods for connecting multiple points into a spline.

3 METHODS AND RESULTS

The following section details how we use the above mathematical analysis to create a polygon and ultimately a spline from the provided track and offset data. The code is provided in the appendix.

In order to create the configuration space, we needed to first take in the track and offset data. Using the track data, we created a simple polygon that fit to all points on the track. Next, in order to create the configuration space, we started by creating a simple polygon that was offset from track polygon. The offset for each cone (or track data point) varies by the uncertainty in the cones position. Applying a variable offset can be difficult; we tried two different approaches to solve this problem. For our first approach we tried to offset the midpoint of an edge. As a result, we got a new point that was defined as the anchor point of the ”new” edge with a direction same or parallel to the original edge. The same procedure was run in a FOR-loop for all the other points of the track. Each intersection of two adjacent edge, creates a new vertex of the offset polygon. The theory is presented in Fig. 1.

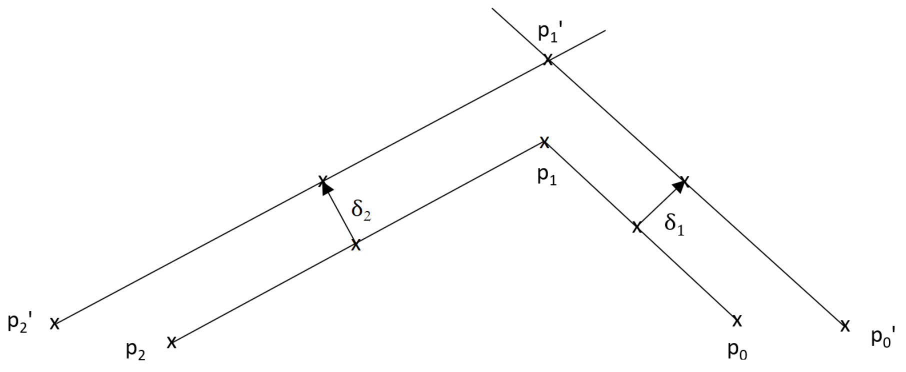

Figure 1: Theory for the first approach

Once we found the new vertices, we used the Python module SciPy to create a continuous spline between all the new vertices. This module is talked about in detail in the Modules Used section. This method resulted in path shown in Fig. 2.

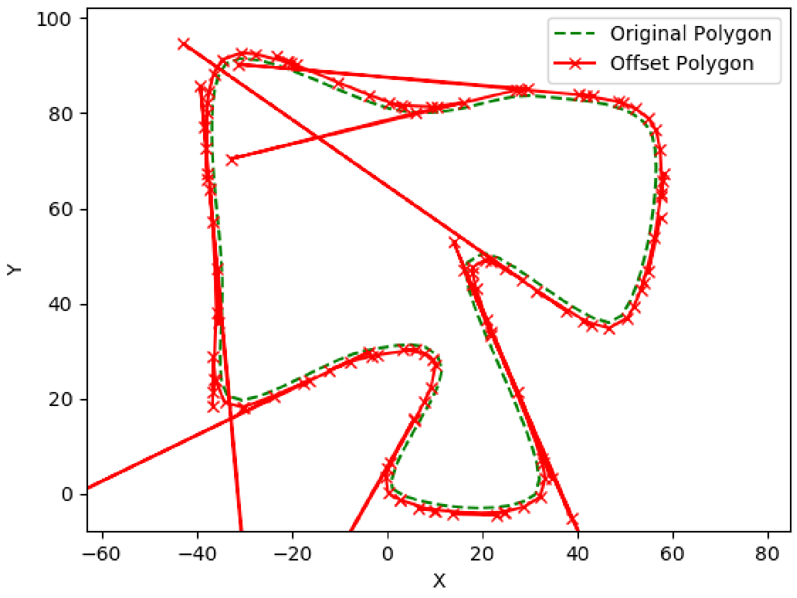

Figure 2: Offset race track - first approach

The problem with this approach is that some intersections seemed to be out of the expected range. For this method, we were following the work did for another project (Homework 3, question 2), where we had to offset a convex polygon with a constant offset. The differences between the two projects are:

- the race track could also be concave
- the offset value varies
- the race track should be a curve as the output

After some analysis, we determined this code does not work for our new problem due to the variable offset. This affects the finding of the intersections or rather the finding of the new vertices.

For our second approach, we applied the offset to the midpoint between vertex pairs instead of offsetting from the vertex itself. We offset from the midpoint by multiplying the unit normal vector with the offset scalar δ. This procedure is also done in a FOR-loop and then all new vertices are connected. The theory is shown in Fig. 3 and Fig. 4 shows the result for this code.

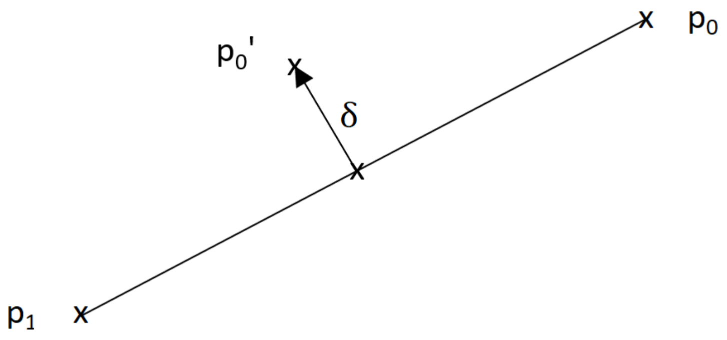

Figure 3: Theory for the second approach

The offset values in Fig. 4 have been exaggerated to show the affect of the variability on the resulting offset polygon. This midpoint method does lead to our offset track consisting of one less point than the original track but this does not seem to cause any major issues, likely because the number of track points is very large.

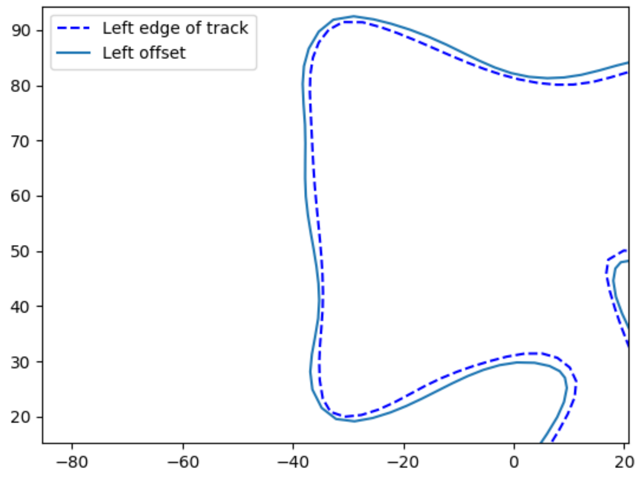

Figure 4: Offset race track - second approach

Lastly, we applied our final approach to the inside and outside track. The result we obtained is presented in Fig. 5.

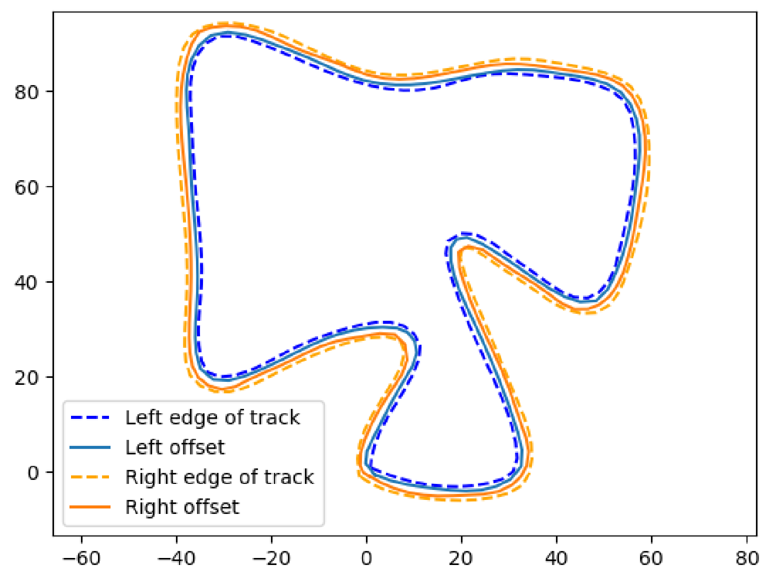

Figure 5: Final approach applied to the inside and outside track

4 Modules Used

The algorithm we created relies on multiple Python modules to perform the necessary calculations.

- Numpy
- Shapely
- SciPy
- Matplotlib

The Numpy module was used to create arrays and perform basic mathematical calculations quickly. Shapely is a computational geometry module we used to create the polygonal representation of the track and configuration space. It is able to transform an array of 2D points into a polygon which can then be manipulated using different operations within the module. From the SciPy module, we used the interpolate package to take the polygonal data from shapely and fit a spline to it. In order to achieve smooth and continuous splines, we used the interpolate.splprep from SciPy(6). This SciPy package can also take in a smoothness value as an input. This smoothness value is a float that is internally used to create a more smooth interpolation of the data points. Finally, we used the Matplotlib module to plot our resulting configuration space.

5 FUTURE WORKS

The algorithm we created when through a few iterations to end up with the version presented in this paper but there are still improvements that can be made. The primary improvement that needs to be made is how the algorithm handles sharp corners. There are a few occasions where using the midpoint approach can lead to the configuration space cutting a corner, such as in Fig. 6, resulting in a small portion of the configuration space that is outside the boundaries of the track. While this area is very small, it is possible that this error could lead the race car on a path that takes it off the track during a tight corner. This is not ideal and will be our main focus going forward. We would additionally like to improve the resolution of the data points, allowing for even smoother curves and transitions between corners. This is likely to improve how the algorithm handles sharp corners as well.

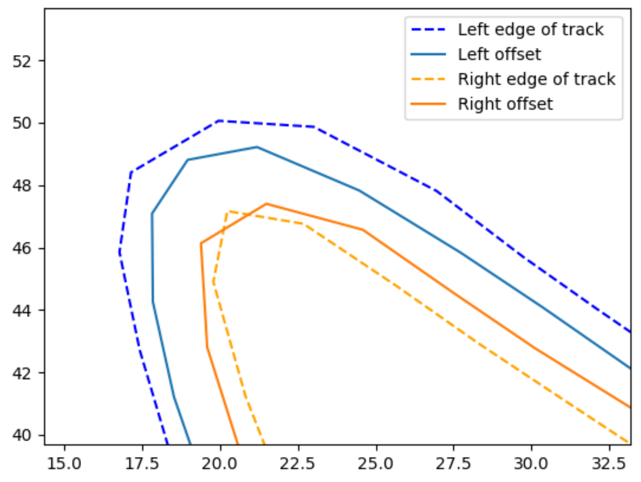

Figure 6: The right offset spline cutting through the right edge of the track

6 CONCLUSIONS

Creating a configuration space is an important first step to keeping an autonomous race car on the track and racing toward the goal. Our configuration space planner uses multiple Python modules to assist in the application of polygons and B-splines to the data of an example race track. By first turning the track into a polygon, we are able to find the midpoints between vertices and apply an appropriate offset at each specific location. These offsets are what help keep the car a safe distance from the edges of the track. By fitting a B-spline to the vertices of the track polygon and the offset polygon, we are able to provide a smooth, continuous configuration space in which the race car can safely operate at all times.

APPENDIX

This appendix contains the Python 3.7 code written for this project.

```
# Nicole Guymer and Niklas Delboi

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import shapely.geometry as shp
from scipy.interpolate import splprep, splev, splrep


def PolyOffset(points, offset):

        # Create a Polygon from the given points
        OGpoly = shp.Polygon(points)

        # Get all the vertices of the offset polygon
        # Define the sizes of some variables that are used in the for loop
        edges = np.zeros((len(points),2))               # Edges of the given polygon
        normals = np.zeros((len(points),2))             # Normal vectors to each edge
        unit_normals = np.zeros((len(points),2))        # Unit normal vectors
        midpoints = np.zeros((len(points),2))           # Midpoint of each edge
        midpoints_offset = np.zeros((len(points),2))    # Midpoint of the offset
                                                          polygon´s edges
        vert_offset = np.zeros((len(points),2))         # Vertices of the offset polygon

        for i in range(len(points)-1):
                edges[i] = np.subtract(points[i+1],points[i])
                normals[i] = (edges[i,1],-edges[i,0])
                mag2 = np.sqrt(normals[i,0]**2+normals[i,1]**2)
                unit_normals[i] = normals[i]/mag2

                # find the midpoint by the half length of each edge
                midpoints[i] = points[i]+0.5*edges[i]

                # get the offset midpoints with the offset as a scalar (delta) and the
                # unit normal vectors
                midpoints_offset[i] = midpoints[i]+offset[i]*unit_normals[i]

        # Do all the same steps for the last edge from the "last point" to the "first
          point" of the given polygon
        edges[len(points)-1] = np.subtract(points[0],points[len(points)-1])
        normals[len(points)-1] = (edges[len(points)-1,1],-edges[len(points)-1,0])
        mag2 = np.sqrt(normals[len(points)-1,0]**2+normals[len(points)-1,1]**2)
        unit_normals[len(points)-1] = normals[len(points)-1]/mag2
        midpoints[len(points)-1] = points[len(points)-1]+0.5*edges[len(points)-1]
        midpoints_offset[len(points)-1] =
        midpoints[len(points)-1]+offset[len(points)-1]*unit_normals[len(points)-1]

        # Define the new polygons
        OffsetPoly = shp.Polygon(midpoints_offset)

        # Turn polygon points into numpy arrays for plotting
        OGpolypts = np.array(OGpoly.exterior)
        OffsetPolypts = np.array(OffsetPoly.exterior)

        # Return the results
        return OGpolypts, OffsetPolypts
```

```
if __name__ == '__main__':

        # Read in the example track information
        wb = np.load('track2.npy')

        # Seperate the inside and outside edges of the track
        wb0 = wb[0]
        wb1 = wb[1]

        # Close the loop by connecting the last point to the first point
        inside = wb0
        outside = wb1

        # Make random set of test offsets
        offsets_inside = np.ones(len(wb0)) # Don't include the repeated start point
        for i in range(len(offsets_inside)):
                offsets_inside[i] = np.random.uniform(low = 0.5, high = 1.25)

        offsets_outside = np.ones(len(wb1))
        for i in range(len(offsets_outside)):
                offsets_outside[i] = np.random.uniform(low = -1.25, high = -0.5)

        inside_track, in_offset = PolyOffset(inside, offsets_inside)
        outside_track, out_offset = PolyOffset(outside, offsets_outside)

        # Spline the points
        x = (inside_track[:,0])
        y = (inside_track[:,1])
        tck, u = splprep([x,y], s = 5)
        new_points = splev(u,tck)

        i = in_offset[:,0]
        j = in_offset[:,1]
        tck1, u1 = splprep([i,j], s = 5)
        new_points1 = splev(u1,tck1)
        plt.plot(new_points[0], new_points[1], '--' ,color = 'blue', label = 'Left edge
                 of track' )
        plt.plot(new_points1[0], new_points1[1], label = 'Left offset')

        x = (outside_track[:,0])
        y = (outside_track[:,1])
        tck, u = splprep([x,y], s = 5)
        new_points = splev(u,tck)

        i = out_offset[:,0]
        j = out_offset[:,1]
        tck1, u1 = splprep([i,j], s = 5)
        new_points1 = splev(u1,tck1)

        # Plot points
        plt.plot(new_points[0], new_points[1], '--' ,color = 'orange', label = 'Right
                 edge of track')
        plt.plot(new_points1[0], new_points1[1], label = 'Right offset')
        plt.axis('equal')
        plt.legend()
        plt.show()
```

References

[1] S. International, “Student events.”

[2] H. Choset, “Robotics motion planning: Configuration space.” University Lecture.

[3] MegunoLINK, “Three methods to filter noisy arduino measurments.”

[4] MathIsFun.com, “Polygons.”

[5] S. J. Janke, *Mathematical Structures of Computer Graphics.* John Woley Sons, Incorporated, 2014.

[6] T. S. community, “scipy.interpolate.splprep.”