Obstacle Avoidance

Melissa Cruz, Yuhong Kan, Maxx Wilson

Collision Geometry and Vehicle Dynamics

The maximum radius arc on the car is traced by the front outer corner of the frame. All calculations include a safety margin to extend the effective size of the vehicle when evaluating collisions. This maximum radius is represented by the equation below, and can be derived using the Pythagorean theorem given the car dimensions, shown in Figure 1 in the Appendix.

$$r_{max} = \sqrt{\left(r + \frac{1}{2}W + m\right)^2 + \left(\frac{1}{2}(L + B) + m\right)^2}$$

The minimum radius arc of the car is traced by the point where the rear axle intersects the inner edge of the vehicle frame. This minimum radius is represented by the equation below, as well as by Figure 2 in the Appendix.

$$r_{min} = r - \frac{1}{2}W - m$$

The “boundary radius”, representing the radius to the front inner corner of the car, is represented by the equation below and by Figure 3 in the Appendix. This was derived similarly to the maximum radius and differentiates front collisions from side collisions.

$$r_{boundary} = \sqrt{\left(r - \frac{1}{2}W - m\right)^2 + \left(\frac{1}{2}(L + B) + m\right)^2}$$

The three radii bound four regions, shown in Figure 4 of the Appendix. Points between the minimum radius arc and the boundary radius arc will result in side collisions. Points between the boundary radius arc and max radius arc will hit the front of the car. No other points will collide with the vehicle. Calculating the radius to the obstacle point allows for the collision region to be determined trivially. Interestingly, there is no condition in which a point will collide with the outside of the car while driving forward given the vehicle’s geometry and dynamics.

The free path length the car can travel before colliding with a point p is described by the following equation:

$$L = r(\alpha - \beta)$$

Where alpha is the angle between the rear axle and the obstacle point, and beta is the angle between the rear axle and the point of collision. The point of collision on the car frame can be derived based on geometry for front and side collisions.

$$\alpha = tan^{-1}\left(\frac{P_{p,y} - P_{o,y}}{P_{p,x} - P_{o,x}}\right) \qquad \beta_{front} = tan^{-1}\left(\frac{0.5*(L+B)+m}{\sqrt{r_p^2 - \left(\frac{1}{2}(L+B)+m\right)^2}}\right) \qquad \beta_{side} = tan^{-1}\left(\frac{\sqrt{r_p^2 - r_{min}^2}}{r_{min}}\right)$$

The minimum stopping distance of the car for a certain velocity is given by the equation below, assuming a known maximum deceleration, a.

$$X_{min} = \frac{V^2}{2a}$$

Algorithm Description

The obstacle avoidance algorithm developed consists of seven primary steps which transform and predict future vehicle state according to system latency, generate and score a number of path options, and then command the vehicle along the path according to optimal control theory.

Algorithm Steps:

1. Forward predict odometry using system latency and buffer of past commands
2. Using the predicted odometry, transform the point cloud to the future vehicle frame to generate commands at the time they will take effect
3. Generate a range of possible paths between the curvature bounds and according to a curvature resolution parameter
4. For each path option, and for each point in the point cloud:
    a. Skip points for which collision is impossible, including points behind the car, away from the direction of turning (i.e. to the left when turning right), or beyond the maximum free path length.
    b. Calculate the free path length to each possible obstacle point
    c. Trim the free path length according to the goal point
    d. Calculate the clearance of the path
5. Select the best path from the possible options according to a scoring function
6. Publish curvature/velocity commands using Time Optimal Control along the chosen path
7. Store the vehicle commands for future use with latency compensation

Latency Compensation:

1. At the beginning of every loop, calculate the time where actuation commands will occur, equal to the current time plus the estimated actuation latency
2. Clear the vehicle commands vector up to the time before the oldest sensor message
3. If there is new odometry, reset the state prediction with the updated sensor message
4. Find the index of the velocity command sent directly before the odometry message
5. Integrate the odometry data to the future actuation time using the velocity command buffer and the actuation latency
6. Integrate the vehicle state between the point cloud timestamp and the odometry timestamp to find the point cloud at the odometry timestamp
7. Combine transforms from the point cloud message to odometry message and the estimated future vehicle odometry to transform the point cloud to actuation time
8. Continue with algorithm using latency adjusted point cloud

Clearance Calculation:

1. Set clearance equal to the maximum clearance

2. For every point (x,y) in the point cloud:
    a. If the point (x,y) is inside the cone defined by the center of rotation (0, 1/path.curvature), car, and the obstacle point
        i. radius_to_point = (point - center of rotation).norm()
        ii. radius = (center of rotation).norm()
        iii. clearance = min(clearance, abs(radius_to_point - radius))
3. Return the clearance

Time Optimal Control:

1. Calculate the minimum stopping distance at the current velocity
2. If the free path is smaller than minimum stopping distance, command zero velocity, otherwise command the maximum velocity

Algorithmic Parameters (Tuning/Impact)

Our obstacle avoidance algorithms use seven primary tunable parameters. The maximum path length parameter limits the length of any given path option, preventing the path length from dominating other factors in the object function in open environments. This was set to 7 meters based on trial-and-error in the 1st floor GDC environment.

The curvature resolution parameter determines the number of curves between the min and max steering angles that will be evaluated. Increasing the curvature resolution also increases system latency due to increased computational requirements (more paths to search). Having estimated an overall system latency on the order of 200 milliseconds, the final curvature resolution was set to 0.02, limiting added latency to ~25 milliseconds. Increases in this value yield decreasing utility past a certain point due to limited point cloud resolution.

The safe stopping distance parameter is used in time optimal control to add a safety factor distance between where the car comes to rest and the detected obstacle. This was set to 5cm for low speeds, and 10cm for higher speeds to ensure safe operation.

The clearance factor is the maximum possible clearance of a given path and bounds the collision search region past the outer edge of the vehicle. This parameter limits the clearance from dominating other factors in the objective function in certain environments. It also reduces obstacle search computation by ignoring points farther than one clearance factor from the outer edge of the car. A 10 meter clearance is no more useful than a 1 meter clearance as long as the car can safely pass.

The clearance objective gain parameter denotes the relative importance of the clearance of a chosen path in the objective function. This is tunable to prioritize clearance over other objectives. Given the typical magnitude difference between a free path length and its clearance, this parameter was set between 20.0 and 40.0. The team printed the values of the three objective function parameters to the console and evaluated their contributions in different situations to tune the relative gains.

The distance to goal objective gain denotes the relative importance of the closest distance between the goal point and a path in the objective function. This parameter was set to -0.1, where the negative sign rewards paths that deviate less from the target point. Given that the goal point was simply to drive forward, this parameter was typically dominated by the path

length and clearance objectives. When a global path is integrated, this parameter will become more important.

The system latency was estimated by commanding a sinusoid to the vehicle velocity, tracking the odometry over time, and calculating the phase difference between the two datasets. System latency was estimated to be around 200 milliseconds.

Algorithm Results

The vehicle performs successful obstacle avoidance in cluttered and open environments between speeds of 1-2m/s. When encountering obstacles that cannot be avoided, the vehicle comes to a stop before collision occurs. The algorithm runs with approximate execution latency less than 1/8th of the estimated system latency, ensuring suitable performance. Latency compensation techniques allow the vehicle to perform effective obstacle avoidance and safe stopping up to a speed of 2 m/s. Tunable objective parameters allow for different algorithmic configurations that can prioritize safer paths with high clearance, or tighter paths with less margin for error.

To acknowledge project shortcomings, the car sometimes experiences small oscillations in the forward state prediction when located around the minimum stopping distance away from an impassable obstacle. Since this was a non-crucial edge case, we did not take steps to remedy this. The small oscillations are likely consequences from imprecise estimates of vehicle dynamics or system latency. Multiple safety factors within the algorithm helped to diminish uncertainties and account for edge cases such as this.

What challenges did you encounter, and how did you overcome them?

Latency compensation was difficult to debug and validate as its contribution is subtle, albeit crucial at high speed. Visualization and methodical software design was a key component to debugging this part of the system.

Managing the time complexity of the curvature search algorithm also required careful balance, where each additional curve required another pass over the point cloud. We were able to reduce the number of points searched by passing each point through different checks with increasing computational requirements. Many points could be immediately excluded based on their X or Y position and insight into vehicle kinematics.

How could you further improve upon your results?

Software optimization would decrease overall system latency. This could include further exclusion of unnecessary points and the replacement of computationally expensive functions like inverse tangent with fast approximations. Improved visualization for the latency compensation and vehicle state transformations would make it easier to debug and improve.

The current iteration of our algorithm does not include functionality to back out of a corner when a continuous escape path is no longer available. This functionality would make environmental exploration more robust.

There was discussion of a non-linear objective function that would penalize smaller clearances heavily while limiting reward for excess clearance. This could improve vehicle

performance in tighter environments without sacrificing safety. This would reduce the number of algorithmic parameters as arbitrary limits for path length and clearance would not be necessary.

A more accurate numerical integration method for forward predicting the state of the car and point cloud could be implemented, such as the Runge-Kutta method. This would improve the accuracy of the car for larger latency as the estimates would not diverge as quickly.

Contributions

Melissa Cruz worked on:

- Estimating system latency experimentally
- Tracking vehicle commands up to the oldest sensor message
- Estimating acceleration based on the current and commanded velocities
- Integrating vehicle state between discrete timestamps
- Transforming sensor data to the future time where control efforts take effect
- Visualizing the projected future vehicle state for debugging

Maxx Wilson worked on:

- Conceptualizing the discrete integration process for latency compensation
- Transforming point cloud from polar to cartesian coordinate systems
- Calculating curvature options based on the vehicle's goal point/curvature
- Computationally cheap methods to pre-filter points before expensive collision search
- Determining whether a point will collide with the front or side, or if it will clear
- Calculating free path length from front and side collisions

Yuhong Kan worked on:

- 1D Time Optimal Control for vehicle commands
- Calculating the distance to goal for a given path option
- Developing scoring function to select a path from a number of path options
- Computationally cheap methods to pre-filter points before expensive collision search
- Calculating minimum clearance for a given path option
- Visualization tools for obstacle avoidance debugging

Github Repository

https://github.com/MaxxWilson/cs393r_starter

Algorithm Performance Videos

https://photos.app.goo.gl/YncJf5R7gJW5KXS98

Appendix

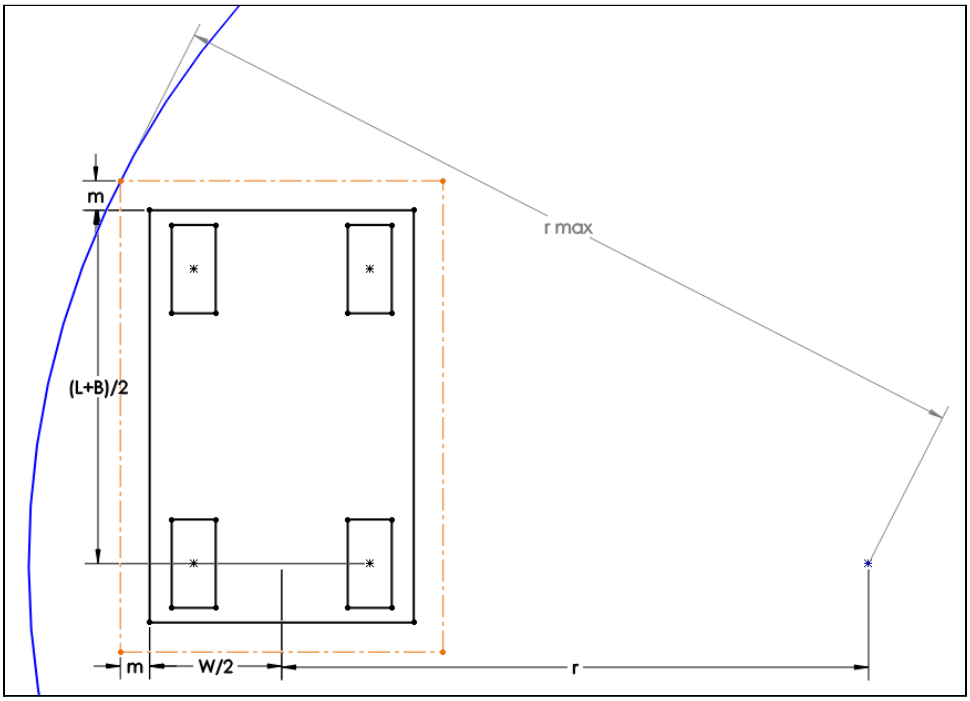

Figure 1

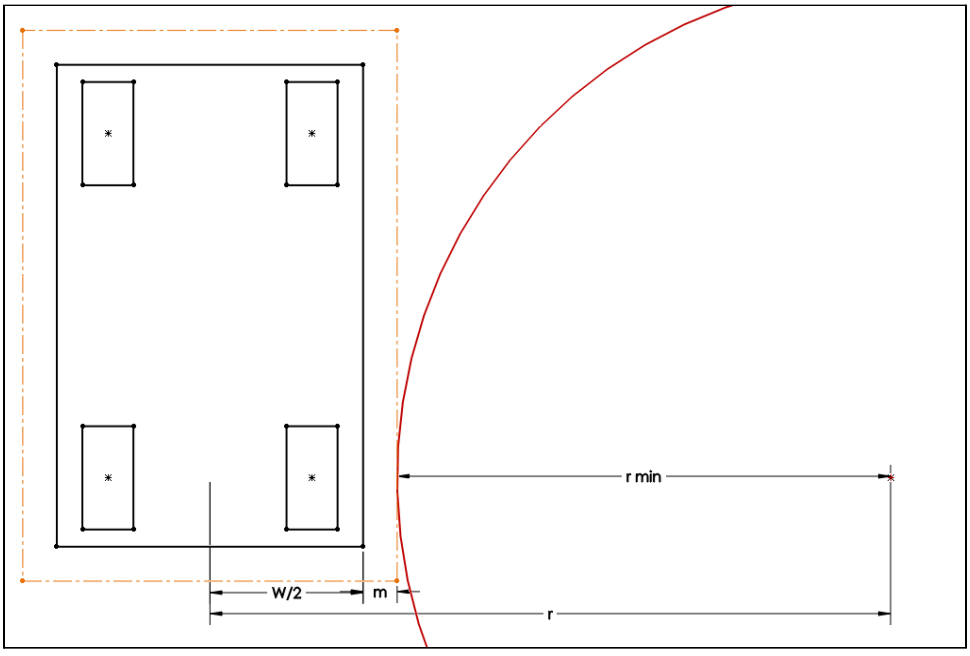

Figure 2

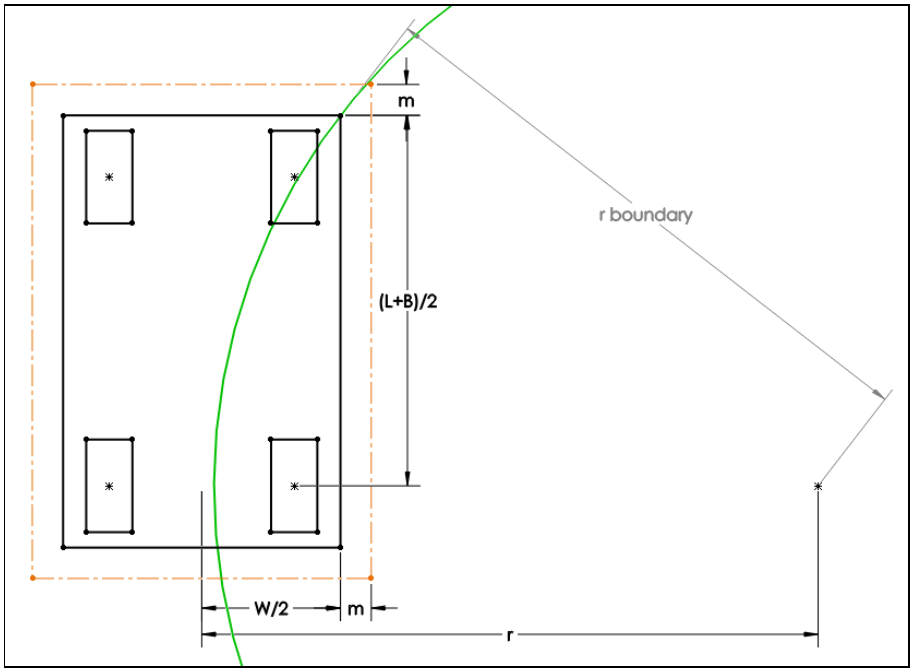

Figure 3

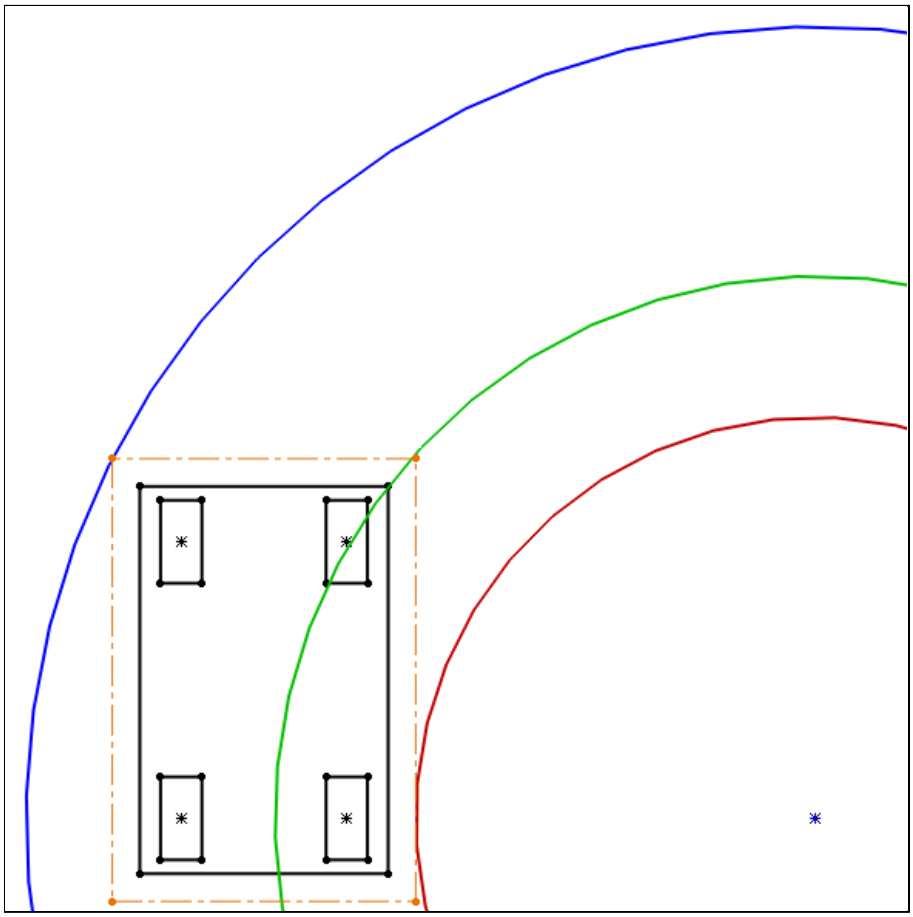

Figure 4