Eric Cox
ericpaulcox@gmail.com

Technical Overview

Introduction

Many path-planning approaches use a robot's configuration space, which defines all possible vehicle locations within its environment, as the basis for path-planning. An example of this is the costmap2D ROS plugin, which inflates obstacles in an occupancy grid according to the robot's dimensions. The ROS navigation stack uses this plugin to identify obstacle-free areas in which to generate feasible paths.

The problem with using solely this approach to obstacle avoidance is that the vehicle's environmental representation is commonly a function of SLAM, which tends to be both computationally costly and thus relatively slow. This can be detrimental to collision-free navigation if the vehicle is traveling faster than the SLAM algorithm's update rate, as obstacles might not be added to SLAM's map in time for the vehicle to avoid a collision. Additionally, if its odometry is not excellent, the vehicle is capable of drifting during navigation. Depending both on how rapidly the vehicle drifts and how rapidly the SLAM process corrects for this drift, the vehicle is capable of colliding with static obstacles while "thinking" it is following an obstacle-free path.

The ROS Navigation stack does a good job compensating for SLAM's relatively low update rate by introducing a second occupancy layer, referred to as the local costmap. This costmap layer is updated at a faster rate than the SLAM process and uses sensor measurements directly (such as LiDAR scans or Kinect pointclouds) to update the configuration space, and consequently, the robot's path. As a result, the path-planner is capable of generating paths which permit navigation in regions where obstacles are directly sensed, but which have not yet been included in the occupancy grid.

Nevertheless, the problem remains that there is no mechanism to actively prevent the vehicle from drifting too close to obstacles. Even with the local costmap layer implemented correctly, the vehicle can get "stuck" in inflated space and the algorithm will fail to produce a path, or the vehicle can collide with static obstacles in the absence of excellent odometry.

In the absence of existing approaches in ROS to actively modify a velocity request such that the vehicle is forced clear of obstacles, a new approach has been developed to act as a supplement to a general path-planning algorithm.

Overview

The active obstacle avoidance node was developed with quadrotors in mind, and is intended to aid a path-planning algorithm. The path-planning algorithm is expected to make intelligent decisions concerning the vehicle's trajectory. The active obstacle avoidance node is intended to run at a higher rate than the path-planner and make corrections to velocity requests generated by the path-planning algorithm.

The algorithm acts as a velocity filter, modifying velocity requests sent from a path-planning algorithm as a function of obstacles sensed relative to the vehicle. As such, the algorithm's performance is independent of the accuracy of the vehicle's localization and is instead dependent on the ability of the vehicle to accurately detect obstacles and execute velocity requests. This decouples the performance of the obstacle avoidance algorithm from any factors which might affect the localization of the vehicle, increasing robustness of the navigation process.

To not collide, the obstacle avoidance algorithm determines the set of acceptable velocities, which includes all quadrotor velocities which will not result in a collision with static obstacles. If the original velocity request is a member of the set of acceptable velocities, then the algorithm will simply serve as a

Eric Cox
ericpaulcox@gmail.com

relay of the original request to the quadrotor. Otherwise, the algorithm will override the request and instead opt for an acceptable velocity most similar to the original request. This process is discussed in more detail in the following sections.

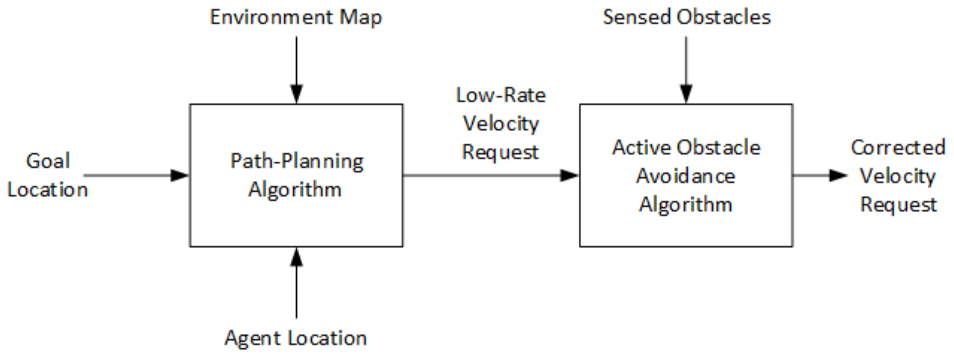

Figure 1. Implementation of the Active Obstacle Avoidance algorithm in the context of a path-planner.

Instantaneous Velocity in the Direction of an Obstacle

To avoid collisions, restrictions are placed on the instantaneous velocity of the quadrotor in the direction of obstacles. These instantaneous velocities, if executed by the quadrotor, would directly modify the distance between the quadrotor and the obstacle. No restrictions are placed on velocity components orthogonal to the direction of the obstacle, since they have no instantaneous bearing on the distance between the quadrotor and an obstacle. The instantaneous velocity of the quadrotor in the direction of a given obstacle is derived below.

First, the distance between the vehicle and the obstacle in a vehicle-fixed coordinate frame is calculated by Pythagorus' theorem:

$$r_o = \sqrt{x^2 + y^2} \tag{1}$$

where r_0 is the direct distance to the obstacle, x is the x-component of distance in the quadrotor's vehicle-fixed frame, and y is the y-component of the obstacle in the quadrotor's vehicle-fixed frame.

Differentiating distance over time yields the instantaneous velocity of the vehicle in the direction of the obstacle, which is given by

$$v_o = \frac{x\dot{x} + y\dot{y}}{\sqrt{x^2 + y^2}} \tag{2}$$

where $\dot{x}$ and $\dot{y}$ represent components of quadrotor velocity in the vehicle-fixed frame. In terms of the angular orientation of the obstacle relative to the vehicle-fixed frame, this equation can be written as

$$v_o = \dot{x}\cos\theta_o + \dot{y}\sin\theta_o \tag{3}$$

where θ_o is the direction of the obstacle with respect to the quadrotor.

Additionally, the velocity of the vehicle must satisfy the following equation, which relates the polar direction of vehicle velocity, θ_v, to Cartesian velocity:

$$\dot{x}\sin(\theta_v) = \dot{y}\cos(\theta_v) \tag{4}$$

Eric Cox
ericpaulcox@gmail.com

Combining Equations 3 and 4, the instantaneous velocity toward an obstacle can be related to the vehicle velocity in polar coordinates:

$$v_v = v_o * \sec(\theta_v - \theta_o) \tag{5}$$

where v_v is the speed of the vehicle, and θ_v is the direction of speed.

Selecting Instantaneous Velocity Constraints

Equation 5 relates the vehicle velocity to the speed in the direction of an obstacle. To ensure that vehicle does not approach an obstacle too rapidly, it is necessary to define a restriction on the speed towards the obstacle, v_o, as a function of distance to the obstacle, r_o. Such a restriction might be given as

$$v_o \leq f(r_o) \tag{6}$$

In other words, the instantaneous velocity of the vehicle towards an obstacle should be less than (or equal to) some function of distance to that obstacle. With this generic restriction implemented, the vehicle's speed in is in turn restricted as a function of the vehicle's course and the relative angle of the obstacle:

$$v_v \leq f(r_o) * \sec(\theta_v - \theta_o) \tag{7}$$

Tuning the Obstacle Avoidance Algorithm

The velocity restriction should be selected with care, as it impacts the control structure of the vehicle. The restriction should be chosen so that system remains stable. If a quadrotor controller which uses velocity to control position has already been established, then the process of choosing a suitable restriction is straightforward.

To illustrate this fact, suppose that the vehicle receives a velocity request from a path-planner which is greater than the maximum velocity permitted by the velocity restriction. In this case, the velocity request which will be sent to the quadrotor will be exactly equal to the velocity restriction:

$$v_v = f(r_o) * \sec(\theta_v - \theta_o) \tag{8}$$

Further assume that the vehicle is traveling directly towards an obstacle, (i.e., $\theta_v = \theta_0$). This results in maximum velocity being exactly equal to the restriction:

$$v_v = f(r_o) \tag{9}$$

The case of the vehicle traveling directly towards an obstacle with a large velocity request can be considered a worst-case scenario, as the velocity restriction will have the largest impact on the velocity request.

If the obstacle is static, then the velocity restriction can be written as a first-order differential equation:

$$\dot{r}_0 = f(r_o) \tag{10}$$

With this realization, the selection for $f(r_o)$ adds an outer control loop to the existing velocity controller (assuming the obstacle in question is static). This control loop is equivalent to that of a position controller. If there is knowledge of both vehicle dynamics and the velocity control structure, the velocity restriction can be selected to ensure stability in a worst-case scenario.

Eric Cox
ericpaulcox@gmail.com

If the control structure of the quadrotor is unknown, then a velocity restriction must be determined experimentally.

Defining Acceptable Velocity Space

The selection of velocity restriction due to a single static obstacle has been examined. In the case of multiple obstacles, the vehicle's acceptable velocity space is given by the most constrictive combination of velocity restrictions. For a given direction, the most restrictive velocity limitation is the minimum bound of all velocity restrictions in that direction.

If the maximum permissible velocity in a given direction due to obstacle *i* is denoted by R_i, where

$$R_i(\theta_v) = f(r_i) * \sec(\theta_v - \theta_i) \tag{12}$$

then the total restriction on velocity in a given direction due to all obstacles is given by

$$\max\big(v_v(\theta_v)\big) = \min\big(R_1(\theta_v), R_2(\theta_v), \dots, R_n(\theta_v)\big) \tag{13}$$

This functions defines the acceptable velocity space for the vehicle.

Sampling the Acceptable Velocity Space

Equation 13, which is a linear programming problem, is complex and expensive to calculate when there are many obstacles. However, the acceptable velocity space can be sampled by evaluating Equation 13 at discrete intervals over θ_v, then interpolating if necessary. This is a conservative approach to estimating the velocity configuration space; while the sampled directions will exactly equal Equation 13, the interpolation between endpoints will yield a result less than or equal to the actual value of the velocity bound. In other words, while the approximation of the velocity bound might result in the vehicle moving toward an obstacle slower than required by Equation 13, it will never result in the vehicle moving toward an obstacle faster than the restriction.

If the sampling interval is a factor of the angular resolution of the measurement sensor (such as a laser scanner), then the nonlinear portion of Equation 8 can be stored in a lookup table (with the index corresponding to the size of the angular increment). This reduces the computational cost of the algorithm the expense of slightly increased memory usage.

Eric Cox
ericpaulcox@gmail.com

ROS Documentation

Overview

The obstacle avoidance node is implemented as a C++/ROS node. The current version subscribes to 2D laser scan topics in ROS (such as those produced by the Hokuyo Lidar). The node also subscribes to a velocity request topic and publishes a modified velocity request topic.

The node does not currently support subscriptions to additional ROS topics, but can be easily modified to incorporate other sensors or locations of known obstacles.

To install, place source code in the catkin workspace, and catkin install. Instructions for installing ROS packaged in general can be found on the ROS website, www.ros.org. The node has been tested successfully with ROS Indigo on Ubuntu 14.04.

Usage

As the control structure of the quadrotor was not known during the algorithm development, a generic linear piecewise function was implemented to define the vehicle's velocity restriction. The piecewise function was selected to decrease the speed of the vehicle as it approaches some user-defined buffer distance from the obstacle, and actively push the vehicle away if it exceeds the buffer distance. The piecewise function is shown below:

$$f(r_o) \leq \begin{cases} v_m\,, & r_o \geq r_p \\ \dfrac{r_o - r_a}{r_p - r_a} v_m, & r_p > r_o \geq r_a \\ \dfrac{r_o - r_a}{r_a - r_r} v_p, & r_a > r_o \geq r_r \\ v_p, & r_r > r_o \end{cases} \tag{11}$$

Parameters are used in the piecewise function to restrict the response of the vehicle are listed below.

r_o — The distance from the obstacle to the vehicle.

r_p — The distance from the obstacle at which the vehicle speed begins to decrease.

r_a — The distance from the obstacle at which the vehicle should come to a stop.

r_r — The distance from the obstacle at which a collision occurs (distance to center of vehicle).

v_m — The maximum velocity that can be requested by the algorithm.

v_p — The maximum speed at which the vehicle can be "pushed" away from an obstacle

The piecewise continuous function is plotted in Figure 2.

Eric Cox
ericpaulcox@gmail.com

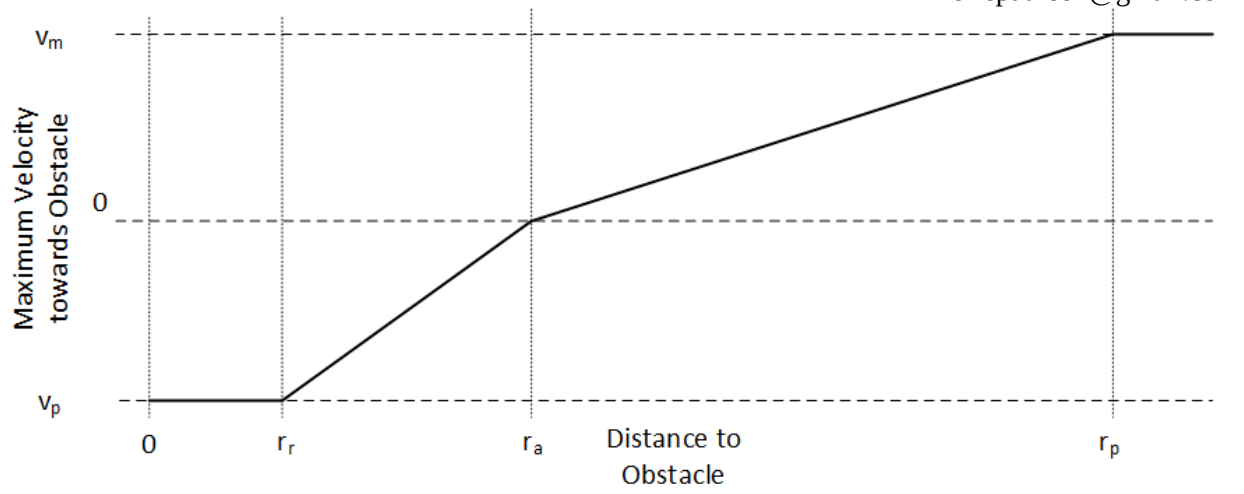

Figure 2. Maximum velocity toward obstacle as a function of distance to obstacle

Parameters for each vehicle should be independently obtained and validated.

Subscriptions

- cmd_vel – A velocity request from a path-planner or other source
- scan – a laser scan topic representing local obstacles relative to the vehicle

Publications

- safe_vel – the modified velocity topic
- dumb_markers – a marker topic visualizing the subsampled velocity space
- robot_radius – a polygon topic representing the robot footprint
- active_radius – a polygon topic representing the distance from an obstacle at which distance the instantaneous velocity toward the obstacle should be zero.
- passive_radius – a polygon topic representing the distance from an obstacle causing a reduction in max velocity toward the obstacle

Parameters

Velocity shaping:

- passive_radius – distance (m) of the passive radius (denoted r_p in overview)
- active_radius – distance (m) from an obstacle at which the vehicle should come to a stop
- robot_radius – distance (m) from an obstacle at which a collision will occur
- max_pos_vel – the maximum velocity that the vehicle should execute
- max_neg_vel – the maximum velocity at which the vehicle can be “pushed” away from an obstacle

Velocity Space Sampling

- num_meas_rays – the number of velocity-sampling intervals to use for approximating the velocity configuration space

Visualization

- publish_profiles – publish visualization topics for RViz