Global Dynamic Window Approach Algorithm

Elizabeth Adelaide

March 29, 2017

Introduction

Navigation for autonomous mobile robots is a difficult problem. One approach is using a global dynamic window approach (GDWA). The algorithm's goal is to navigate a course based upon sensor input. GDWA uses several factors to base its navigation on. These factors are continuously updated, and then the weighted average is taken. GDWA has the advantage of being able to balance several aspects of navigation, including current heading, distance to goal and obstacles.

Algorithm

The algorithm is based upon finding the optimal velocity within a range of values based on the current velocity. This range of values is the window of potential velocity values. The robot needs to have a system to receive inputs, through sensors. GDWA has the advantage of being able to incorporate several types of sensor information. While more accurate information about the velocity can be found using feedback systems, the nature of GDWA does not require exact knowledge about the velocity of the robot.

The velocity of the robot can be described using several variables. The velocity can be described as a velocity vector ($\bar{V}$). The velocity can be described as a tuple of speed (*v)* and rotational speed (ω). This coordinate system is most appropriate for wheeled robots that are most directly controlled based on the speed and the angular speed. Other robots might use cartesian coordinates (v_x, v_y, v_z). The robot also records a position vector ($\bar{X}$). For a wheeled robot this will be the *x* and *y* coordinate, as well as the angular position α .

GDWA calculates a value for a range of values within the window. The window is centered on the current velocity vector. The general form of the algorithm is:

$$GDWA(\bar{V}_i) = \sum_j K_j F_j(\bar{V}_i)$$

where $\bar{V}_i$ is a given velocity vector within the window range. $F_j(\bar{V})$ is a normalized function on the interval [-1, 1]. Each function is weighted by a constant K_j . The weights determine the relative importance of each function. The function should be created such that it has a highest value at the most favorable velocity. At each time step, GDWA calculates each velocity vector within the dynamic

window range. The vector with the highest value is chosen as the next velocity. The window is updated, and the new velocity is calculated. *Figure 1* shows the algorithm.

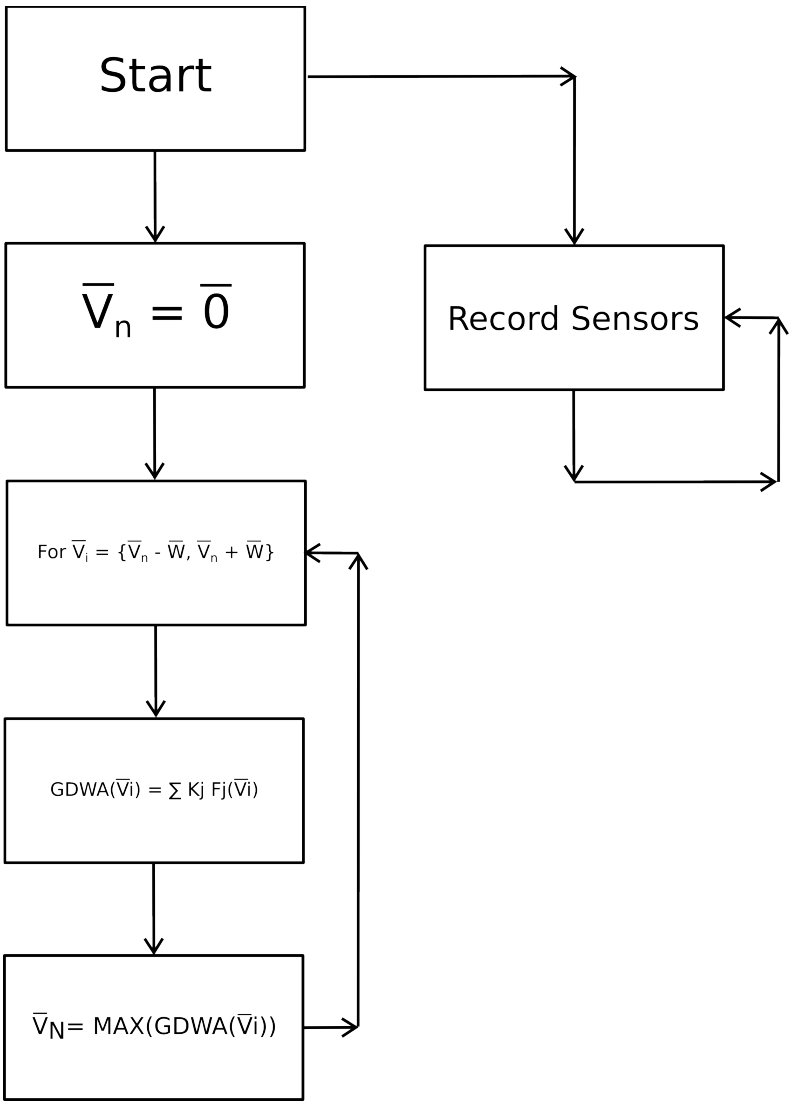

Figure 1: GDWA Algorithm

The algorithm has several performance parameters. The sensors can be checked asynchronously, they can be recorded at different time intervals than the GDWA calculations. This can be an advantage when working with slow or complicated sensors. The size of the window and the coarseness of the values checked are major impacts on the performance. Smaller windows and coarse values will lower the time of each step. Larger windows and fine values will increase the time of each step. Smaller times between each step will increase the ability of the robot to respond to changes in the sensor data. Smaller times and larger windows will increase the ability of the robot to accelerate quickly. Coarse values will decrease the time step, however it may cause sharp or unstable changes in velocity. The physical ability of the robot to accelerate of the robot is another factor to consider. An algorithm that calculates accelerations too fast or slow will be prone to errors and offset errors.

Example Algorithm

The general algorithm can be used for a variety of purposes. It is helpful to see a specific application of the algorithm. Consider a wheeled robot with speed *v* and angular speed *ω*. The GDWA calculation is then:

$$GDWA(v_i, \omega_i) = \sum_j K_j F_j(v_i, \omega_i)$$

The engineer's challenge is to first decide what functions should be used to calculate the new velocity vector. These functions should be based on the problem at hand. As an example the robot needs to navigate to a goal in a course with obstacles. The robot knows the approximate location of the goal, but can only sense the goal when it is close to the goal. The robot should avoid obstacles, but should reach the goal as fast as possible. The functions should then direct the robot towards the goal, avoid obstacles, and to increase the velocity to fastest value. GDWA allows functions to be easily added and removed.

For this example a few functions will be used. A basic approach would be to write four functions. The first function chooses values which move the robot towards the location of the goal ($goal(v, \omega)$). The second avoids obstacles ($avoid(v, \omega)$). The third favors the highest velocity ($veloc(v, \omega)$). These three functions will navigate the robot to the goal. The GDWA calculation can be written as:

$$GDWA(v_i, \omega_i) = A\, goal(v_i, \omega_i) + B\, avoid(v, \omega) + C\, veloc(v_i, \omega_i)$$

Each of these functions can be written through several approaches. An advantage of GDWA is that they can be changed without changing the rest of the program. A disadvantage is that the behavior of the robot is not immediately predictable. GDWA requires testing to ensure that there is predictable behavior. Additionally, it is good practice to consider special cases where GDWA should no longer be used. For example if the robot is trapped at zero velocity for a long period of time, the algorithm should switch to a different algorithm.

Goal Function

While each of these functions can be written in several ways, the most basic implementation is illustrative. In this problem there are two problems. The first is when the exact location of the goal is unknown. The second is when the exact location of the goal is known. The exact location is known when the robot is within sensor range of the goal.

Before the robot detects the goal, it can calculate the approximate distance to the goal given its current position:

$$d(x, y) = \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2}$$

Where *d* is the distance, *x* and *y* are the position of the robot, and x_{goal} and y_{goal} is the approximate location of the goal. For a given velocity vector, the function calculates the position of the

robot one time step later:

$$x' = x + \frac{v}{\omega}(\cos(\omega \Delta t + \alpha) - \cos(\alpha))$$
$$y' = y + \frac{v}{\omega}(\sin(\omega \Delta t + \alpha) - \sin(\alpha))$$

where Δt is one time step and α is the current angular position of the robot. The function then calculates the change in distance, Δd :

$$\Delta d = d(x', y') - d(x, y)$$

The value is normalized by the maximum distance moved during one time-step, $v \Delta t$:

$$goal(v_i, \omega_i) = \frac{\Delta d}{v \Delta t}$$

The time-step should be chosen such that $v \Delta t$ is smaller than the range of the sensor that will detect the exact location of the goal. Otherwise, the robot may behave unpredictably close to the goal. A more advanced approach would be dynamically adjust the time step based on the distance of the goal.

Avoid Function

The avoid function can be implemented in several ways. A simple approach would be to value any velocity vectors that move go towards sensors that are not detecting any values, or detecting values at the longest distance. Another approach which can take advantage of several types of sensors is the use of a local map. A local map is a map of places where the robot has detected an obstacle. The map is populated each time a sensor detects an object. The local map follows the robot, and is updated as the robot moves.

The local map is populated at each sensor detection. The map is an grid of values which represents the certainty that there is an obstacle at a given point. When a sensor detects an object at a given distance, the object's position is approximated on the local map. The object is assumed to be directly in front of the sensor. Most sensors have some angular spread, however this is neglected. The value at this position is increased by *p*. The values that are between the object's approximate position and the sensor are decreased by *n*. It is very unlikely for there to be an object between a sensor and the object that was actually detected. Therefore the probability on the local map is decreased. The local map initializes as all zero values. Negative values indicate that there is no obstacle at that position, positive values indicate there is likely an object. *Figure 3* shows a sample object being detected.

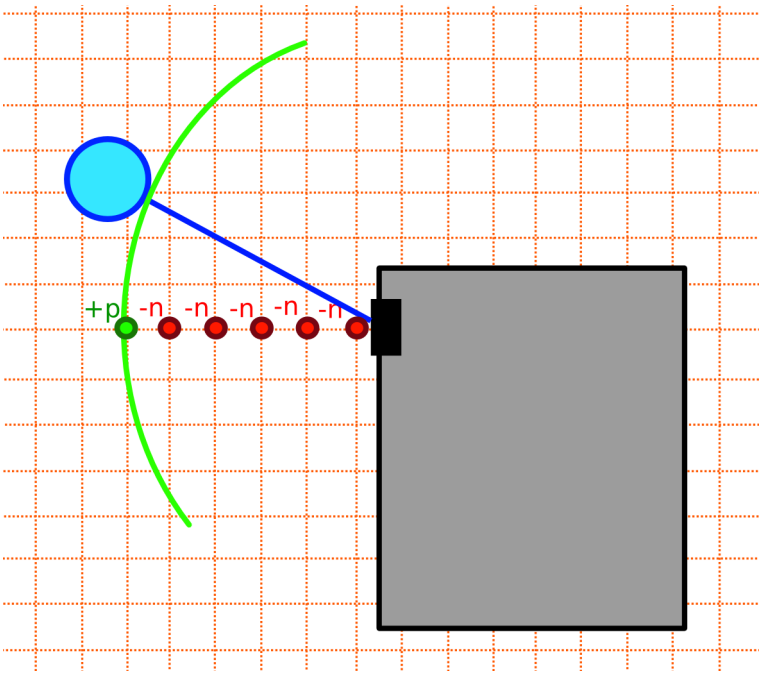

Figure 3: Detection of an Obstacle

The *avoid* function calculates the position of the robot at several time steps Δt_i if the robot occupies a space on the local map that is likely to contain an obstacle, the function returns a -1. If there are no obstacles along the path the function returns 1. This approach is useful for a local map which is relatively small and dense. This is designed to avoid imminent collision.

The local map offers a few advantages. The map can be updated asynchronously, allowing the GDWA loop to operate separately from the detection loop. Updating the map can also be computationally intensive, however these calculations can be done without directly impacting the performance of the GDWA algorithm. This approach also can lower noise from sensors, an outlier in the sensor will be removed the next time the map is updated.

Velocity Function

The velocity function is a straight forward function. The function favors greater magnitude velocities. This function forces the robot to accelerate, and can also prevent the robot from becoming stuck. The function is:

$$veloc\left(v_i, \omega_i\right) = \frac{|v_i|}{v_{max}}$$

Where v_{max} is the maximum velocity of the robot. The function can be made non-linear to improve the robots performance at special cases. The function can be altered to be more sensitive at small values, which can helpful to prevent the robot from being stuck. The function can also favor rapid acceleration after a given velocity.

Conclusion

GDWA is a flexible approach to controlling an autonomous robot. It requires a careful design process based on the physical attributes of the robot and the problem presented. GDWA is a modular approach, which can use a variety of functions to affect the behavior of the robot. Using a goal, velocity and avoid function will give the robot functionality. Additional functions can be added for more efficient navigation and more complicated behavior.