Turrets and deadzones

A visual look at a turret's control flow

Evan Pratten

2020-08-31

Overview

A turret is generally a simple mechanism to program. In its simplest form, the goal of a turret is to point at things. Common uses of turrets in robotics are for: aiming launchers at targets, rotating a camera to look around a space, and swiveling manipulators (like an arm). While the systems built on top of, and around a turret may be very complex, a turret's control flow is one of the simplest there is (figure 1).

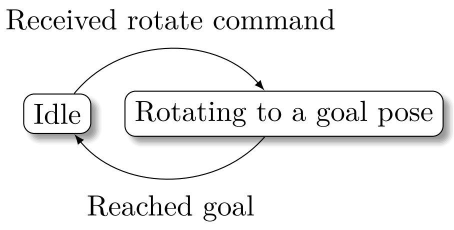

Figure 1: A simple turret's control flow

A turret's position can be expressed as a *rotation*. Rotations are essentially a circle, where a point on the circle can be expressed as an angle in degrees ranging from $-180°$ to $+180°$ (see figure 2)

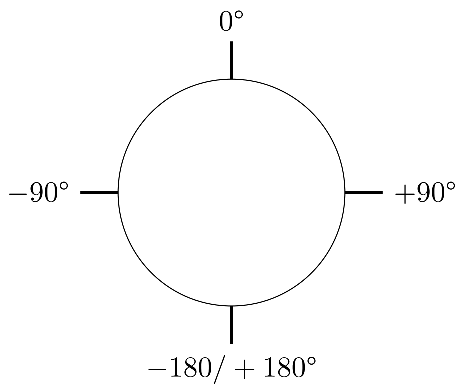

Figure 2: A visualization of a rotation

Infinite-rotation turrets vs. turrets with deadzones

An infinite-rotation turret is a mechanism that can rotate in any direction an infinite number of times. These turrets do not need any kind of path planning. Calculating the difference between their current rotational pose, and the goal pose will result in the number of degrees the turret must rotate from its current position to reach the goal. Unfortunately is is uncommon to find a turret that allows truly infinite rotation. While some turrets may allow $\pm 1080°$ rotation (not covered in this paper), almost all turrets have a deadzone.

Deadzones are the limits of a turrets rotation. These are generally implemented to stop a turret from ripping apart any sensor cables that may be attached to it. With a deadzone, the previous control implementation of rotating to the difference between two poses will not always work. What if the shortest path between two poses crosses of the deadzone? Figure 3 contains a visualization of a deadzone on a rotation, where d_{min} is the counter-clockwise-most boundary, and d_{max} is the clockwise-most boundary of the deadzone.

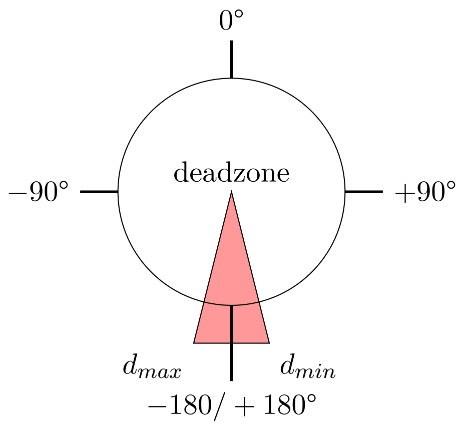

Figure 3: A visualization of a rotation and a deadzone

A very naive solution to this problem is to make turret rotate to the right until it either reaches the goal pose, or bumps into the deadzone. If the turret bumps into the deadzone, it then starts rotating left until it hits the goal pose. This solution causes the turret to sweep right and left until it finds its goal (see figure 4).

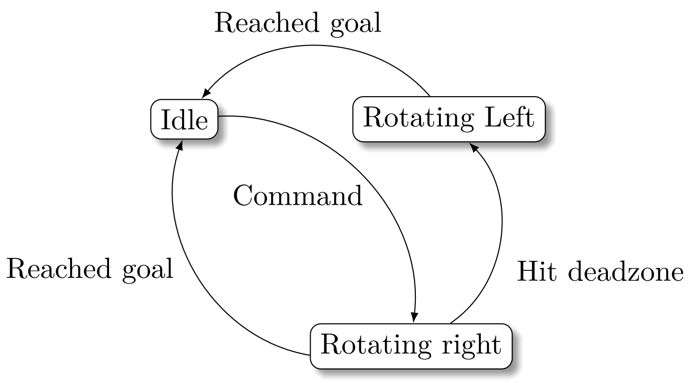

Figure 4: Naive deadzone avoidance control flow

While this works, it is the least efficient possible solution. What if the goal is 1° to the left of the turret's current position? Instead of just moving left, the turret will have to move all the way to the right, then come back to the goal. This is a huge waste of time.

Optimal turret control

Recently, I was tasked with designing a solution for determining the shortest path around a deadzone, and noticed that by shifting a rotation to a new 0-based numbering system (figure 5), this problem can be solved very easily. To visually simplify this process, I have “unwrapped” the rotation circle into a line (figure 6).

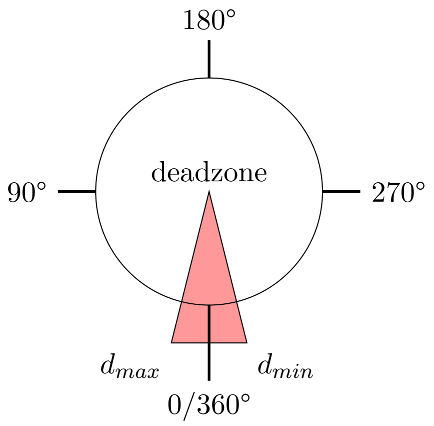

Figure 5: Rotation with deadzone crossing 0/360°

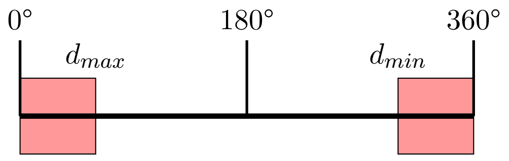

Figure 6: Rotation on a line with deadzone crossing 0/360°

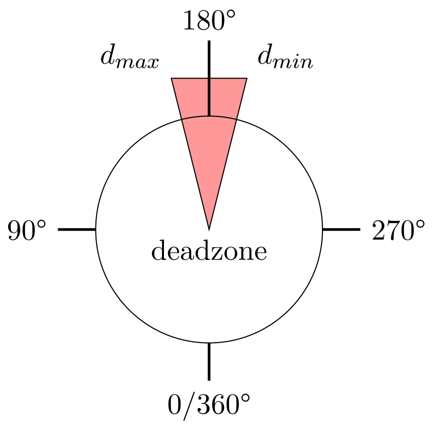

Figure 7: Rotation with deadzone crossing 180°

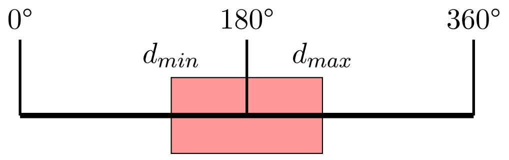

Figure 8: Rotation on a line with deadzone crossing 180°

As shown in figures 6 and 8, the deadzone can have one of two properties. It can either form a “wall” on each side of the number line, or form a “block” in the middle.

When the deadzone forms a wall, the turret will only ever have to navigate between two points in the empty space. In this case, the distance the turret needs to travel can always be solved by taking the difference between the current pose (A) and the goal pose (B). Just as long as B is never placed inside a deadzone, this will work perfectly.

In the case that the deadzone forms a block, a little bit of math is required to find the shortest distance between A and B. Even though the difference between 10° and 350° is only 20°, subtracting them will result in the (incorrect) answer of 340°. Luckily, a small set of equations can fix this problem.

$$\begin{aligned} \phi &= abs(A - B) \pmod{360} \\ D &= \text{copysign}(\{\phi > 180 : 360 - \phi, \phi\}, A - B) \end{aligned}$$

Figure 9: Calculating D, the relative distance from A to B